UC San Diego
UC San Diego Electronic Theses and Dissertations

Title
Enabling Technologies and Applications for Networked Airborne Computing

Permalink
https://escholarship.org/uc/item/31m217hs

Author
Wang, Baoqian

Publication Date
2023

Peer reviewed|Thesis/dissertation

eScholarship.org
Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA SAN DIEGO

SAN DIEGO STATE UNIVERSITY

Enabling Technologies and Applications for Networked Airborne Computing

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Engineering Science (Electrical and Computer Engineering)

by

Baoqian Wang

Committee in charge:

University of California San Diego
Professor Nikolay Atanasov, Co-Chair
Professor Jorge Cortes
Professor Yan Wan
Professor Michael Yip

San Diego State University
Professor Junfei Xie, Co-Chair
Professor Jun Chen

2023

Copyright

Baoqian Wang, 2023

All rights reserved.

The dissertation of Baoqian Wang is approved, and it is acceptable in quality and form for publication on microfilm:

____

____

____

____

____
Co-Chair

____
Co-Chair

University of California San Diego

San Diego State University

2023

TABLE OF CONTENTS

Dissertation Approval Page iii

Table of Contents iv

List of Figures viii

List of Tables xiii

Acknowledgements xiv

Vita xvi

Abstract of the Dissertation xviii

Chapter 1 Introduction 1
1.1 Background 1
1.2 Contributions of the Dissertation 3

Chapter 2 Airborne Computing Platform Design 6
2.1 Introduction 6
2.2 Hardware Design for the Airborne Computing Platform 7
2.2.1 Desired Features 8
2.2.2 Single-Board Computer Selection 8
2.2.3 A Prototype 10
2.3 Software Design for the Airborne Computing Platform 13
2.3.1 Background and Related Work 13
2.3.2 Computing Performance 15
2.3.3 Networking Performance 17
2.3.4 Isolation Performance 19
2.3.5 Power Consumption 21
2.3.6 Discussions 22
2.4 Performance of the Airborne Computing Platform 24
2.4.1 OpenDroneMap 24
2.4.2 Real-Time Object Detection 26
2.5 Conclusion 28
2.6 Acknowledgement 28

Chapter 3 Batch-Processing Based Coded Comput-ing for Static Networked Airborne Computing Systems 29
3.1 Introduction 29
3.2 Related Work 33
3.3 System Models 35
3.3.1 Computing System 35

3.3.2 Computing Schemes . . . 35
3.3.3 Problem Formulation . . . 37
3.4 Main Results . . . 39
3.4.1 Notations for Asymptotic Analysis . . . 39
3.4.2 A Simplified Formulation . . . 39
3.4.3 A Two-Step Alternative Formulation . . . 40
3.4.4 Solution to the Two-Step Alternative Problem . . . 41
3.4.5 Optimality Analysis . . . 44
3.4.6 Analysis of the Impact of Parameter p . . . 45
3.4.7 Comparison with HCMM . . . 46
3.5 Simulation Studies . . . 47
3.5.1 Simulation Settings . . . 47
3.5.2 Parameter Impact Analysis . . . 48
3.5.3 Comparative Performance Studies . . . 53
3.6 Experiments on the Amazon EC2 Computing Cluster . . . 54
3.6.1 Experiment Settings . . . 54
3.6.2 Parameter Estimation . . . 55
3.6.3 Experimental Results . . . 56
3.7 Conclusion . . . 61
3.8 Acknowledgement . . . 62

Chapter 4 Learning and Batch-Processing Based Coded Computation for Mobile Networked Airborne Computing Systems . . . 63
4.1 Introduction . . . 63
4.2 Related Work . . . 65
4.2.1 Networked Airborne Computing . . . 65
4.2.2 UAV-assisted Mobile Edge Computing . . . 66
4.2.3 DRL-based UAV-Assisted Networks . . . 66
4.2.4 Coded Distributed Computing . . . 67
4.3 NAC System . . . 68
4.4 Dynamic Batch-Processing Based Coded Computation Framework . . . 69
4.4.1 D-BPCC Framework . . . 69
4.4.2 Problem Formulation . . . 71
4.5 DRL-based Solution to $\mathscr{P}_1$ . . . 75
4.5.1 RL based Formulation for $\mathscr{P}_1$ . . . 75
4.5.2 Deterministic Policy Gradient Method . . . 77
4.5.3 Deep Neural Network based Function Representation . . . 78
4.5.4 Training DRL Agent . . . 79
4.5.5 Convergence and Complexity Analysis . . . 82
4.6 DRL-based Solution to $\mathscr{P}_2$ . . . 82
4.6.1 RL based Formulation for $\mathscr{P}_2$ . . . 82
4.6.2 Solution to $\mathscr{P}_2$ . . . 84
4.7 Simulation Studies . . . 85
4.7.1 Simulator Design . . . 85

4.7.2 Benchmarks ... 88
4.7.3 Evaluation of Solution to $\mathscr{P}_1$ ... 90
4.7.4 Evaluation of Solution to $\mathscr{P}_2$ ... 94
4.8 Conclusion ... 97
4.9 Acknowledgement ... 98

Chapter 5 Coded Distributed Multi-Agent Reinforcement Learning with One-hop Neighbors ... 99
5.1 Introduction ... 99
5.2 Related Work ... 102
5.2.1 Multi-Agent Reinforcement Learning ... 102
5.2.2 Distributed and Parallel Architectures for RL and MARL ... 103
5.2.3 Coded Distributed Computing ... 104
5.3 Background ... 104
5.4 Problem Statement ... 105
5.5 Distributed multi-Agent Reinforcement Learning with One-hop Neighbors ... 107
5.6 Coded Distributed Learning Architecture ... 110
5.6.1 Coded Distributed Learning Architecture ... 111
5.6.2 Assessment of Gradient Estimator ... 113
5.7 Assignment Matrix Construction ... 114
5.7.1 Uncoded Assignment Scheme ... 114
5.7.2 Coded Assignment Schemes ... 116
5.8 Experiments ... 119
5.8.1 Performance of DARL1N ... 119
5.8.2 Performance of Coded Distributed Learning Architecture ... 127
5.9 Conclusion ... 135
5.10 Acknowledgement ... 136

Chapter 6 Simulator and Testbed Design and Implementation ... 137
6.1 Introduction ... 137
6.2 Computing Model ... 139
6.3 Simulator Design ... 140
6.3.1 UAV Hardware Module ... 140
6.3.2 Controller Module ... 141
6.3.3 Visualization Module ... 142
6.3.4 Wireless Communication Module ... 142
6.3.5 Computing Module ... 143
6.3.6 ROS Topics ... 144
6.4 Hardware Testbed Design ... 145
6.4.1 Computing ... 145
6.4.2 Communication ... 145
6.4.3 Flight Control ... 146
6.4.4 Power Management ... 146
6.5 Simulation Studies ... 147

6.5.1 Simulation Configurations . . . 148
6.5.2 Simulation Results . . . 148
6.6 Real Flight Tests . . . 150
6.6.1 Experiment Configurations . . . 151
6.6.2 Experiment Results . . . 152
6.7 Discussions and Conclusions . . . 154
6.8 Acknowledgement . . . 155

Chapter 7 Conclusion and Future Work . . . 156
7.1 Conclusion . . . 156
7.2 Future Work . . . 159

Appendix A Proofs of Chapter 3 . . . 160
A.1 Proof of Lemma 1 . . . 160
A.2 Proof of Lemma 2 . . . 162
A.3 Proof of Theorem 4 . . . 170
A.4 Proof of Theorem 6 . . . 173
A.5 Proof of Corollary 6.1 . . . 175
A.6 Proof of Theorem 7 . . . 175

Appendix B Proofs of Chapter 5 . . . 176
B.1 Proof of Lemma 9 . . . 176
B.2 Proof of Proposition 1 . . . 177
B.3 Proof of Proposition 3 . . . 178

Bibliography . . . 179

LIST OF FIGURES

Figure 1.1. Illustration of Networked Airborne Computing (NAC) paradigm. 2

Figure 2.1. a) A new Jetson TX2 carrier board and b) Jetson TX2 carrier board with the processor. 11

Figure 2.2. A prototype of the airborne computing platform. 12

Figure 2.3. Execution time of the SC application implemented on CPU using increasing number of threads. The number of input points in the SC application is set to a) 10000 and b) 900000. 16

Figure 2.4. Execution time of the SC application implemented on GPU and CPU of six threads. The number of input points in the SC application is set to a) 10000 and b) 900000. 16

Figure 2.5. Bandwidth of the communication link a) between airborne computing platform and ThinkPad laptop, b) between two airborne computing platforms, and c) between two airborne computing platforms. 18

Figure 2.6. Power consumption of the airborne computing platform before and after running the SC application. 22

Figure 2.7. Response of the host OS when the a) VM or b) container is attacked. 24

Figure 2.8. Execution time of the a) image resizing and b) 3-D model reconstruction functions in OpenDroneMap to process a UAV image in different virtualization environments. 25

Figure 2.9. The 3-D geographical model generated from 41 UAV images using the 3-D model reconstruction function in OpenDroneMap. 26

Figure 2.10. Execution time of two applications a) when running separately and when running simultaneously in Jetson TX2 without virtualization, b) when running simultaneously in Jetson TX2 of three different virtualization setups. 27

Figure 2.11. A UAV image a) before and b) after applying the DNN based object detection. 27

Figure 3.1. Illustration of the a) uncoded and b) coded computation to perform matrix multiplication with $n = 2$. The numbers marked in green describe the computation procedures. 31

Figure 3.2. The approximated execution time τ^* of BPCC at different values of a) p_1, when $p_j = 1, \forall j \in [N] \setminus \{1\}$, and b) p, when $p_i = p, \forall i \in [N]$, in different scenarios. 49

Figure 3.3. The value of a) load ℓ_1^* and b) total load $q = \sum_{i=1}^{N} \ell_i^*$ at different values of p, when $p_i = p, \forall i \in [N]$, in different scenarios. 49

Figure 3.4. The mean execution time of BPCC at different values of a) p_1, when $p_j = 1$, $\forall j \in [N] \setminus 1$, and b) p, when $p_i = p, \forall i \in [N]$, in different scenarios. 51

Figure 3.5. The approximation error of τ^* at different values of N with $r = 100N + 10000$. 52

Figure 3.6. Relative change of the mean execution time when a) the straggling parameters μ_i and b) the shift parameters α_i suffer from different degrees of deviation from their true values in different scenarios. 53

Figure 3.7. a) Comparison of the mean execution time of different schemes in different scenarios. b) The average total number of rows of inner product results received by the master node over time for different schemes in Scenario 2. 53

Figure 3.8. The CDF of the processing time of an Amazon EC2 t2.xlarge instance for computing a task with $r = 500$. 56

Figure 3.9. a) The mean execution time of different schemes in different scenarios at the presence of unexpected stragglers with finite delay. The b) success rate and c) mean execution time of different schemes in different scenarios at the presence of unexpected stragglers with infinite delay. 58

Figure 3.10. The average total number of rows of inner product results received by the master node over time for different schemes in Scenario 4 at the presence of unexpected stragglers with a) finite and b) infinite delay. 59

Figure 3.11. a) The mean execution time of different schemes in Scenario 4 when different percentages of unexpected stragglers with finite delay are present. b) The success rate of different schemes in Scenario 4 when different percentages of unexpected stragglers with infinite delay are present. 60

Figure 3.12. The a) mean execution time of different schemes in Scenario 4 when different percentages of unexpected stragglers with infinite delay are present. b) The mean execution time of BPCC at different values of p in Scenario 4. . 61

Figure 4.1. Cooperative airborne computing of matrix-vector multiplication tasks under the D-BPCC framework. 70

Figure 4.2. DNN representation of the policy functions for computation load optimization. . . . 79

Figure 4.3. The training process of the DRL-based method for NAC with uncontrollable UAVs. . . . 81

Figure 4.4. DNN representation of policy function μ for joint computation load and UAV mobility optimization. . . . 84

Figure 4.5. a) Minimum task completion time $\alpha l + \xi$ versus task size l. b) CDF of the task completion time of an Amazon EC2 *t2.xlarge* instance for computing $\mathbf{Ax}$ with $l = 500$. . . . 88

Figure 4.6. Training reward of our method for $\mathscr{P}_1$. . . . 91

Figure 4.7. Average task completion times of different methods in different scenarios when a) $\xi = 0$ and b) $\xi = 0.04$. . . . 92

Figure 4.8. Average task completion times of different methods in different scenarios when $\xi = 0$ and computation times have a large variance. . . . 93

Figure 4.9. Average task completion times of different methods in different scenarios when there are a) one and b) two worker nodes leaving the NAC network. 93

Figure 4.10. Success rates of different methods in Scenario 2 ($N = 6$) when an increasing number of worker nodes leave the NAC network. . . . 94

Figure 4.11. Training reward of our method for $\mathscr{P}_2$. . . . 95

Figure 4.12. a) Total cost, b) average task completion times and c) total flight time of different methods in different scenarios. . . . 96

Figure 4.13. Sample trajectories of the UAVs in Scenario 1 by using a) our method with joint optimization; and b) benchmark methods. . . . 97

Figure 4.14. Sample trajectories of the UAVs in Scenario 1 a) when computation tasks are completed; and b) when the whole mission is completed. . . . 97

Figure 5.1. (a) One-hop neighbor transitions from one time step to the next in a d-disk proximity graph; (b) Coded distributed learning architecture. . . . 111

Figure 5.2. Average training time of different methods to run (a) 10 iterations in the Ising Model, (b) 30 iterations in the Food Collection, (c) 30 iterations in the Grassland, and (d) 30 iterations in the Adversarial Battle environments. 121

Figure 5.3. Average total training reward of different methods in the Food Collection environment when there are (a) $M = 12$, (b) $M = 24$ agents. 124

Figure 5.4. Mean and standard deviation of normalized total reward of competing agents trained by different methods in the Adversarial Battle environment with $M = 48$. 125

Figure 5.5. States of a subset of agents during an episode in Adversarial Battle with agents trained by different methods when there are $M = 48$ agents. 127

Figure 5.6. Average total training reward of DARL1N and SAC in the Multi-Access Wireless Communication environment when (a) $z = 2$ (b) $z = 10$. 128

Figure 5.7. Overhead introduced by different agent assignment schemes when there are $M = 12$ agents and $N = 24$ learners. 129

Figure 5.8. Overhead introduced by (a) Random Sparse and (b) LDGM schemes when their parameters take different values. 130

Figure 5.9. Resilience of different agent assignment schemes to stragglers when the straggler probability η increases. 131

Figure 5.10. Resilience of (a) Random Sparse and (b) LDGM schemes when their parameters take different values. 132

Figure 5.11. Average V of different agent assignment schemes calculated using results from different number of learners. 133

Figure 5.12. Average training time of different DARL1N implementations with straggler effect a) $\Delta = 1$ and b) $\Delta = 4$. 134

Figure 6.1. NAC simulator design. 141

Figure 6.2. NAC hardware testbed design. 144

Figure 6.3. Visualization of the simulation environment with Gazebo. 147

Figure 6.4. Visualization of the UAVs' pre-planned waypoints and paths with Rviz. . . 147

Figure 6.5. a) Trajectories of the four UAVs and b) distances between the master UAV and worker UAVs in the simulation. 148

Figure 6.6. a) Throughput between the master and worker UAVs and b) completion time for a single iteration of matrix multiplication in the moving scenario. 149

Figure 6.7. a) Throughput between the master and workers and b) completion time for a single iteration of matrix multiplication in the static scenario. 150

Figure 6.8. Throughput between two UAVs at various distances in simulations. 150

Figure 6.9. Flight test of NAC hardware testbed with three UAVs at the San Diego State University (SDSU) sport field. 151

Figure 6.10. Trajectories of the three UAVs. 151

Figure 6.11. a) Distance and b) throughput between the master UAV and worker UAVs in the moving scenario. 153

Figure 6.12. Throughput between two UAVs at various distances in flight tests. 153

Figure 6.13. a) Time for completing 10 training iterations and b) the associated training cost in the moving scenario. 154

Figure 6.14. a) Throughput between the master UAV and worker UAVs in the static scenario; b) Time for completing 10 training iterations and c) the associated training cost in the static scenario. 154

LIST OF TABLES

Table 2.1. Comparison of different single-board computers 9

Table 2.2. Performance degradation of the well-behaved guest in different stress tests . 20

Table 2.3. Resource usage of a bare VM or container 22

Table 3.1. Execution time of uncoded and coded matrix multiplication with $n = 2$ . . . 32

Table 3.2. Estimated computing parameters of different types of Amazon EC2 instances 57

Table 5.1. Configurations of Amazon EC2 instances 122

Table 5.2. Convergence time and convergence reward of different methods in the Ising Model environment. 123

Table 5.3. Convergence time and convergence reward of different methods in the Food Collection environment. 124

Table 5.4. Convergence time and convergence reward of different methods in the Grassland environment. 126

Table 5.5. Convergence time and convergence reward of different methods in the Adversarial Battle environment. 126

Table 5.6. Convergence time and convergence reward of different DARL1N implementations. 135

Table 5.7. Average V of different DARL1N implementations. 135

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Professor Junfei Xie, for her support and guidance throughout my Ph.D. journey. I appreciate the opportunity Professor Junfei Xie offered me to pursue a Ph.D. degree and conduct research in the Unmanned Aerial Systems area, which I am passionate about. Professor Xie is nice and patient. She guided me on how to do research and help me to make improvements. She not only cares about my academic performance but also my personality development. She always encourages me to overcome difficulties and always keep positive and be confident in myself. I am very lucky to have her as my Ph.D. advisor!

I also would like to give many thanks to Professor Nikolay Atanasov for being my co-advisor. Professor Nikolay Atanasov is an expert in the robotics area. His guidance is valuable in shaping my research ideas and improving the quality of my work. He made the commitment to ensuring that our research meets the highest standards of quality and rigor. He is willing to challenge me and push me beyond my comfort zone, which has helped me to develop my critical thinking skills and to become a more confident and capable researcher.

I am also grateful to other professors, including Professors Jorge Cortes, Michael Yip, Jun Chen, Yan Wan, Shengli Fu, and Kejie Lu for their constructive feedback, stimulating discussions and encouragement. I also appreciate the help and support from my internship mentor Denis Osipychev and manager Chad McFarland from Boeing. They provide valuable feedback and discussion on my research work from the perspective of the industry.

Last but not least, I would like to express my sincere appreciation to my parents and other family members, for their unconditional love and support throughout my academic journey. Their unwavering belief in me has been a constant source of strength and inspiration.

This dissertation is mainly composed of content from following published/submitted papers:

Chapter 2, in part, is a reprint of the published journal: B. Wang, J. Xie, S. Li, Y. Wan, Y. Gu, S. Fu, K. Lu, "Computing in the Air: An Open Airborne Computing Platform", *IET*

Communications, Vol.14, pp. 2410-2419, 2020.

Chapter 3 is a reprint of the published journal: B. Wang, J. Xie, K. Lu, Y. Wan, S. Fu, “On Batch-Processing Based Coded Computing for Heterogeneous Distributed Computing Systems”, *IEEE Transactions on Network Science and Engineering*, Vol.8, pp:2438-2454, 2021.

Chapter 4 is a reprint of the published journal paper: B. Wang, J. Xie, K. Lu, Y. Wan, S. Fu, “Learning and Batch-Processing Based Coded Computation with Mobility Awareness for Networked Airborne Computing”, *IEEE Transactions on Vehicular Technology*, Nov. 2022.

Chapter 5 is based on the published conference proceedings: B. Wang, J. Xie, N. Atanasov, “DARL1N: Distributed multi-Agent Reinforcement Learning with One-hop Neighbors”, *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*; B. Wang, J. Xie, N. Atanasov, “Coding for Distributed Multi-Agent Reinforcement Learning”, *2021 International Conference on Robotics and Automation (ICRA)*, and on the submitted journal paper: B. Wang, J. Xie, N. Atanasov, “Coding for Distributed multi-Agent Reinforcement Learning with One-hop Neighbors”, *IEEE Transactions on Neural Networks and Learning Systems*, Nov. 2022.

Chapter 6, in full, has been submitted for publication of the material as it may appear in: B. Wang, J. Xie, K. Ma, Y. Wan “UAV-based Networked Airborne Computing Simulator and Testbed Design and Implementation”, *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*.

VITA

2013-2017 Bachelor of Engineering., Yangtze University, Wuhan

2017–2019 Master of Science, Texas A&M University, Corpus Christi

2019–2023 Doctor of Philosophy, University of California San Diego/San Diego State University

PUBLICATIONS

Journal Publications

B. Wang, J. Xie, N. Atanasov, "Coding for Distributed multi-Agent Reinforcement Learning", *IEEE Transactions on Neural Networks and Learning Systems*, Nov. 2022 (under review).

B. Wang, J. Xie, K. Lu, Y. Wan, S. Fu, "Learning and Batch-Processing Based Coded Computation with Mobility Awareness for Networked Airborne Computing", *IEEE Transactions on Vehicular Technology*, Nov. 2022.

B. Wang, J. Xie, K. Lu, Y. Wan, S. Fu, "On Batch-Processing Based Coded Computing for Heterogeneous Distributed Computing Systems", *IEEE Transactions on Network Science and Engineering*, Vol.8, pp:2438-2454, 2021.

B. Wang, J. Xie, S. Li, Y. Wan, Y. Gu, S. Fu, K. Lu, "Computing in the Air: An Open Airborne Computing Platform", *IET Communications*, Vol.14, pp. 2410-2419, 2020.

J. Xie, Y. Wan, B. Wang, S. Fu, K. Lu, J. Kim, "A Comprehensive 3-Dimensional Random Mobility Modeling Framework for Airborne Networks", *IEEE Access*, Vol.6, pp. 22849-22862, 2018.

Conference Publications

B. Wang, J. Xie, N. Atanasov, D. Osipychev "Learn and Allocate: Learning-Based Scalable Multi-Robot Task Allocation", *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (under review).

B. Wang, J. Xie, K. Ma, Y. Wan "UAV-based Networked Airborne Computing Simulator and Testbed Design and Implementation", *2023 International Conference on Unmanned Aircraft Systems (ICUAS)* (under review).

B. Wang, J. Xie, N. Atanasov, "DARL1N: Distributed multi-Agent Reinforcement Learning with One-hop Neighbors", *2022 IEEE/RSJ International Conference on Intelligent Robots and*

Systems (IROS).

B. Wang, J. Xie, N. Atanasov, "Coding for Distributed Multi-Agent Reinforcement Learning", *2021 International Conference on Robotics and Automation (ICRA).*

D. Wang, B. Wang, J. Zhang, K. Lu, J. Xie, Y. Wan, S. Fu, "CFL-HC: A Coded Federated Learning Framework for Heterogeneous Computing Scenarios", *2021 IEEE Global Communications Conference (Globecom)*.

B. Zhou, J. Xie, B. Wang, "Dynamic Coded Convolution with Privacy Awareness for Mobile Ad Hoc Computing", *International Conference on Communications (ICC)*, Dec. 2021.

B. Wang, J. Xie, K. Lu, Y. Wan, S. Fu "Multi-Agent Reinforcement Learning Based Coded Computation for Mobile Ad Hoc Computing", *2021 International Conference on Communications (ICC)*.

C. Douma, J. Xie, B. Wang, "Coded Distributed Path Planning for Unmanned Aerial Vehicles", *2021 AIAA Aviation Forum*.

B. Wang, J. Xie, J. Chen, "Data-Driven Multi-UAV Navigation in Large-ScaleDynamic Environment Under Wind Disturbances", *2021 AIAA Scitech Forum*.

B. Wang, J. Xie, Y. Wan, G. A. G. Reyes, L. R. G. Carrilo,"3-D Trajectory Modeling for Unmanned Aerial Vehicles", *2019 AIAA Scitech Forum*.

B. Wang, J. Xie, K. Lu, Y. Wan, "Coding for Heterogeneous UAV-based Networked Airborne Computing", *2021 IEEE Global Communications Conference (Globecom) Workshop*.

B. Wang, J. Xie, S. Li, Y. Wan, S. Fu, K. Lu,"Enabling High-Performance Onboard Computing with Virtualization for Unmanned Aerial Systems", *2018 International Conference on Unmanned Aircraft Systems (ICUAS).*

ABSTRACT OF THE DISSERTATION

Enabling Technologies and Applications for Networked Airborne Computing

by

Baoqian Wang

Doctor of Philosophy in Engineering Science (Electrical and Computer Engineering)

University of California San Diego, 2023
San Diego State University, 2023

Professor Junfei Xie, Co-Chair
Professor Nikolay Atanasov, Co-Chair

Unmanned Aerial Vehicles (UAVs) are widely used in many civilian and military applications such as package delivery, precision agriculture, mobile edge computing, and reconnaissance. In these applications, UAVs often need to perform computationally expensive tasks such as path planning, object detection, or mobile computing services. However, due to the small payload, the amount of computing resources individual UAVs can carry is limited. Although significant advances have been made in improving UAV technologies from aspects such as mechanics, control, communication, and networking, enhancing the onboard computing capacity of UAVs hasn't gained much attention as above mentioned aspects. This dissertation aims to fill this

research gap by exploring Networked Airborne Computing (NAC), a new computing paradigm that aims to achieve high-performance airborne computing via inter-vehicle resource sharing using direct flight-to-flight communication links.

We first investigate how to enhance the onboard computing capacity of individual UAVs in Chapter 2 by designing the onboard hardware and software. As the computing capability of individual UAVs is still limited due to small payload, Chapters 3 and 4 further explore how to leverage resources from neighboring UAVs to enhance a UAV's airborne computing capacity by using distributed computing techniques. Particularly, Chapter 3 investigates the static scenario where UAVs hover in the air while conducting computations. To optimize airborne computing performance, a coded distributed computing framework is introduced. Chapter 4 extends the analysis to dynamic scenarios where UAVs are in motion during computation, and a mobility-aware coded distributed computing framework is proposed to address these scenarios. The computation problem considered in both chapters is fundamental matrix multiplication problem, which serves as building blocks for many other advanced computation problems. In Chapter 5, we shift our attention to a more complicated problem, multi-agent reinforcement learning (MARL), and investigate how to reliably and efficiently train MARL over NAC systems. Finally, Chapter 6 designs and implements a realistic simulator and hardware testbed, and conducts experiments to evaluate the performance of NAC in two computation applications including distributed matrix multiplication and distributed gradient descent. Our experiments offer valuable insights into NAC and provide guidance for future advancements.

Chapter 1

Introduction

This chapter overviews background of Networked Airborne Computing (NAC), discusses related work and motivates our studies. The contributions of this dissertation are then highlighted at then end of this chapter.

1.1 Background

Over the past few years, unmanned aerial vehicles (UAVs) have become increasingly important. On the one hand, a single UAV or a group of them can support many commercial and civilian applications, such as forest-fire detection [1], reconnaissance [2], search and rescue [3], and 3-D mapping [4]. On the other hand, UAV can be connected with many ground-based devices to facilitate more applications, such as providing wireless services to cellular users, increasing connectivity in vehicular networks, delivering medical supplies to disaster areas, and offering computing services to ground users [5, 6, 7]. Among these applications, using UAVs to assist computing has recently drawn a growing interest. The implementation of advanced UAV functions (e.g., path planning, positioning, video processing, flight control) also requires a significant amount of computing resources. However, due to small payload, the computing capability of most existing UAV platforms is very limited. To execute computation-intensive tasks, the existing solutions are to offload these tasks to ground servers or remote clouds, which can, however, incur significant delays or even failures [8].

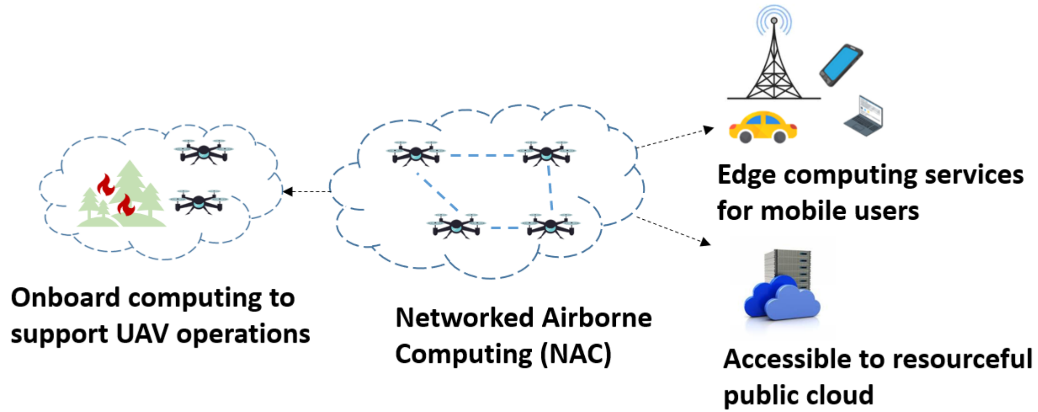


Figure 1.1. Illustration of Networked Airborne Computing (NAC) paradigm.

To address the aforementioned issues, a promising technique is the NAC [9] that can offer advanced onboard airborne computing capabilities. NAC is a new computing paradigm formed by aerial vehicles connected with direct flight-to-flight communication links (see Fig. 1.1). The advantages of UAV-based NAC include low latency, transportability, infrastructure-free, unmanned maneuvering, fast deployment, wide-coverage, and low cost. It can not only enhance UAVs' system performance by allowing advanced algorithms to be implemented onboard of UAVs and hence benefit a wide range of existing UAV applications, but also give rise to a variety of new applications. For example, it can facilitate data collection, processing, and distribution for Internet of Things (IoT) devices, and can function as Mobile Edge Computing (MEC) servers [10] to provide computing services for ground users, etc.

Despite the exciting advantages and broad applications, enabling a UAV-based NAC requires overcoming many technical challenges. For example, when UAVs operate in the complex and uncertain airspace with high mobility, the fast node movement, line-of-sight effect, and node leaving and joining can cause frequent topology changes, link failures, data losses, and task interruptions. Moreover, the various uncertainties (e.g., winds and other vehicles) present in the airspace can disturb the communication among the UAVs, bringing additional challenges for robust computing.

Currently, the research on UAV-based NAC is still in its early stage, and most existing

studies have been focused on the UAV-assisted MEC [11, 12, 13, 14, 15, 16, 17, 18] with a single UAV, which is just one of many possible applications of NAC. Specifically, these studies investigate how to provide the best computing services to ground users, via properly allocating resources and planning UAV trajectories. Moreover, in these studies, the UAVs act alone without collaboration and follow trajectories that are pre-planned. The locations of ground users are assumed to be known and fixed.

Moreover, in existing studies on UAV-based computing [19, 20, 21, 22, 23], performance evaluation was typically conducted through simulations with UAV movement, communication, and computing behaviors described using mathematical models. While mathematical model-based simulations offer the advantage of being inexpensive and easy to deploy, their underlying models, due to simplicity, may not accurately reflect the intricate behavior of real UAV systems. Realistic simulator and hardware testbed for NAC are lacking.

1.2 Contributions of the Dissertation

This dissertation aims to enable Networked Airborne Computing, a new computing paradigm based on UAVs that can enhance computing capabilities of UAVs for advanced applications and provide computing services to users. The main contributions of dissertation are summarized as follows:

Chapter 2 develops a networked airborne computing platform with powerful computing capability to allow computationally expensive applications directly completed onboard. We aim to enhance computing capability of individual UAVs by selecting powerful micro-computer Jetson TX2 as the computing unit and investigating virtualization techniques KVM and Docker for computing resources management, platform security and programmability.

Chapter 3 further improves the computing capability of individual UAVs through leveraging resources from neighboring UAVs by using distributed computing techniques. Particularly, we introduce a novel coded distributed computing (CDC) framework for static and heterogeneous

NAC systems. This framework leverages coding theory to improve computation robustness to uncertain stragglers, which are computing nodes that can fail or delay computing assigned task due to communication bottleneck, system uncertainties and so on. An optimal computation load allocation method called Batch Processing-based Coded Computation (BPCC) is proposed to optimize the load allocation among UAVs. Theoretical analysis reveals the asymptotic optimality of BPCC and the impact of its important parameters. We also prove that it outperforms a state-of-the-art CDC scheme for heterogeneous systems, called Heterogeneous Coded Matrix Multiplication (HCMM)[24, 25].

In Chapter 3, the focus is on static networks with hovering UAVs. In Chapter 4, this work is extended to include mobility of UAVs and explore optimal computation load allocation in mobile networks. Particularly, we examine two realistic scenarios for the formation of NAC networks. The first scenario involves the formation of the NAC system by UAVs operated by different owners in an opportunistic manner, e.g., when cargo drones owned by different companies are serving the same area. In this case, the mobility of the UAVs is uncontrollable, unknown, and can be considered random. The second scenario involves the formation of the NAC system by UAVs operated by the same owner, e.g., in multi-UAV applications like multi-UAV surveillance, search and rescue. In this case, the mobility of the UAVs can be controlled and proactively planned by the owner to facilitate computing. Considering these two formation scenarios, we develop innovative computation schemes to enable efficient, robust, and adaptable cooperative airborne computing in an uncertain, heterogeneous, and dynamic airspace.

The computation problem considered in both Chapter 3 and Chapter 4 is matrix multiplication, which is a building block for many estimation and control algorithms. In Chapter 5, we investigate a more complex problem, i.e., multi-agent reinforcement learning (MARL). We investigate how to train MARL over NAC efficiently and reliably. In particular, we propose a scalable MARL algorithm called Distributed multi-Agent Reinforcement Learning with One-hop Neighbors (DARL1N) that can be trained over a distributed computing architecture. DARL1N reduces the representation complexity of the value and policy functions of each agent in a MARL

problem by disregarding the influence of other agents that are not within one hop of a proximity graph. This model enables highly efficient distributed training, in which a compute node only needs data from an agent it is training and its potential one-hop neighbors. As stragglers in NAC can exist due to communication bottleneck, or software and hardware problems, the training process can be delayed or even fail. To improve the resilience of DARL1N to stragglers common in distributed computing systems, we developed coding schemes that assign each agent to multiple learners. The properties of Maximum Distance Separable (MDS), Random Sparse, Repetition, Low Density Parity Check (LDPC), and Low Density Generator Matrix (LDGM) codes were evaluated.

Chapter 6 addresses the research gap regarding the scarcity of realistic testbeds for NAC research by introducing a realistic simulator and hardware testbed. The simulator was developed using ROS (Robot Operating System) [26] and Gazebo [27]. The hardware testbed comprises multiple UAVs with computing and inter-vehicle resource sharing capabilities. Through simulations and real flight tests, we examine the impact of UAV mobility on NAC by evaluating two computation applications. Our findings provide valuable insights into the challenges of achieving high-performance NAC and provide guidance for future enhancements of the proposed NAC testbed.

Chapter 2

Airborne Computing Platform Design

2.1 Introduction

This chapter aims to design a UAV platform with powerful airborne computing capability to enable NAC. In the literature, many researchers have been working on the design of UAV platforms focusing on control [28], communications [29, 30, 31], networking [32], etc. Nevertheless, we notice that the computation aspect of the UAV platforms has been largely neglected. For instance, most existing UAV platforms have limited computing capability and can perform only essential functionality, such as flight control, image/video capturing, and sensor data collection [33, 34, 35, 36]. Consequently, computation-intensive tasks are often offloaded to the ground stations or to the cloud, which may lead to many issues. For instance, such a computing model may lead to significant transmission delays or failures, and thus cannot support many delay-sensitive applications. Moreover, for many high-bandwidth applications, such as real-time object detection and tracking, such a model requires large communication bandwidths, which may not be feasible in certain scenarios. These issues can be addressed by directly carrying out computation-intensive tasks onboard the UAVs.

Motivated by the aforementioned need, this chapter aims to develop an airborne computing platform with powerful computing capability, broadband communication, flexibility, programmability, and security. To achieve it, we design the platform from both hardware and software aspects described as follows:

1. *Hardware Design.* We first investigate how to design and implement the hardware of the UAV-based airborne computing platform. Specifically, we discuss the desired features for the airborne computing platform, especially according to the unique UAV structure and applications. We then conduct a comprehensive analysis and systematic study on ten state-of-the-art single-board computers regarding the computation performance, power consumption, size, and weight. Based on the thorough comparison, we choose NVIDIA Jetson TX2 [37] as the computing unit for the platform. A prototype is then designed and implemented, which integrates Jetson TX2 for onboard computations, UAV, broadband communication system and networked control system.

2. *Software Design.* With respect to the software of the platform, we aim to design an airborne computing platform with sufficient flexibility and programmability. A key technology to achieve this goal is virtualization, because it can efficiently manage resources, can enable concurrent applications, and can enhance the security of the computing platform. In this chapter, we investigate two key virtualization techniques: (1) virtual machine (VM) using KVM [38] and (2) container using Docker [39]. To understand the impact of virtualization on UAV applications, we conduct extensive experiments to measure the performance of the two virtualization techniques from the aspects crucial for UAV applications, including computing, networking, isolation, power consumption, etc. The performance trade-offs are also discussed. These experiments verify the feasibility of virtualizing UAV and demonstrate the potentials of virtualization in enhancing UAV' onboard computing capability. The insights obtained from the comparison results between KVM and Docker also provide guidelines for the selection of appropriate virtualization techniques for UAV applications.

2.2 Hardware Design for the Airborne Computing Platform

In this section, we investigate the hardware design for UAV high-performance onboard computing. In particular, we first discuss the desired features for the onboard computing hardware.

We then analyze ten state-of-the-art single-board computers and provide guidelines for choosing a suitable single-board computer as the computing unit. A prototype designed based on a selected single-board computer is then described.

2.2.1 Desired Features

Since the onboard computing hardware is carried by a UAV, there are some unique considerations for the selection of a single-board computer. First, the computing hardware should be of light weight and compact size, due to the limited payload and space provided by the UAV. Second, because of the limited power capacity of the UAV-carried batteries, it is preferable to have an efficient power management system to reduce the power consumption. Third, in terms of computing, the hardware should have a powerful CPU, sufficient memory and storage to support most computing needs. Moreover, a powerful GPU is also necessary to enable real-time image processing and deep learning capabilities onboard of UAV. Last but not the least, the single-board computer should have extensive community support, such that developers and users can share experience and seek online support during their system development. In addition, it is also very important to have sufficient open access design documentation and configuration toolboxes.

2.2.2 Single-Board Computer Selection

In the literature, several single-board computers are commonly used for UAV operations, including Raspberry PI [40], Odroid XU [41], Arduino Board [42], Cubieboard [43] and Arndale Board [44], etc. In our study, we do not consider them because they are not powerful enough to fulfill computation-intensive tasks.

Unlike the aforementioned computing devices, more powerful single-board computers have not been fully investigated for UAV. Recently, Shang and Shen [45] investigated the computing power of NVIDIA Jetson TX1 [46] with 4 CPU cores, 256 Maxwell CUDA GPU cores and 4GB memory. Their studies show that NVIDIA Jetson TX1 is still not sufficient enough to achieve real-time 3D reconstruction and mapping using the simultaneous localization

Table 2.1. Comparison of different single-board computers

	CPU	GPU	Memory	Connectivity	Dimension (mm)	Power consumption	OS	Weight	Virtualization support	Storage	Price
Jetson TX2	Denver 2 (2 cores) 2MB Cache, 2GHz + ARM® A57 (4 cores) 2MB Cache, 2GHz	256-core NVIDIA Pascal GPU	8 GB	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth	50×87	7.5W	Linux	85g	Yes	32GB	$400
Jetson AGX Xavier	8-core NVIDIA Carmel Armv8.2 64-bit CPU	512 NVIDIA CUDA cores and 64 Tensor cores	64 GB	10/100/1000 BASE-T Ethernet	100×87	10-30W	Linux	274g	Yes	32GB	$1861
Jetson TX1	ARM Cortex-A57 (4 cores) 2MB L2	256-core NVIDIA Maxwell GPU	4 GB	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth	50×87	10W	Linux	88g	Yes	16GB	$299
Raspberry Pi 4	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8), 1.8GHz	Broadcom VideoCore VI	8 GB	Gigabit Ethernet, 802.11 b/g/n/ac WLAN, Bluetooth	88×58	4-6W	Linux	46g	Yes	microSD	$75
UDOO X86 ULTRA	Intel® Pentium N3710 (4 cores) 2MB Cache, 2.56GHz	Intel® HD Graphics 16 units, 405-700 MHz	8 GB	1 Gigabit Ethernet, M.2 Key E slot for optional Wireless (WiFi+Bluetooth)	120×85	6W	Windows, Linux, Android	117g	Yes	32GB	$267
Intel Aero Compute Board	Intel® Atom^{TM} x7-Z8750 (4 cores) 2MB Cache, 2.56GHz	Intel® HD Graphics 16 units, 405-600 MHz	4 GB	Intel® Dual Band Wireless-AC 8260	$88 \times 63 \times 20$	7.5W	Linux	30g	Yes	32GB	$399
Lattepanda Alpha	Intel® 7th Gen M3-7Y30 (2 cores) 4 MB Cache, 2.60GHz	Intel® HD Graphics 615 300-900MHz	8 GB	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth	$113 \times 80 \times 13.5$	NA	Windows Linux	104g	Yes	64GB	$398
UP Squared	Intel® Apollo Lake (2-4 cores)	Intel® Gen 9 HD with 12 (Celeron) or 18 (Pentium) Execution Units	8 GB	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth	85.6×90	NA	Windows, Linux, Android	NA	Yes	128GB	$399
DJI Manifold	ARM Cortex-A15 (4 cores)	192-core NVIDIA CUDA GPU	2 GB	10/100/1000BASE-T Ethernet	$110 \times 110 \times 26$	5-15W	Linux	197g	Yes	16GB	$499
HiKey 960	ARM Cortex-A73 (4 cores) +Cortex A53 (4 cores)	ARM Mali G71 MP8	4 GB	WiFi, Bluetooth 4.1	$85 \times 55 \times 9$	NA	Linux AOSP	60g	Yes	32GB	$249
Rock 960	ARM Cortex-A72 (2 cores) Cortex A53 (4 cores)	ARM Mali T860 MP4	4 GB	WLAN 802.11 ac/a/b/g/n, Bluetooth 4.2	$85 \times 54 \times 11$	NA	Linux AOSP	120g	Yes	32GB	$139

and mapping algorithm.

To select a suitable single-board computer to enable UAV high-performance onboard computing, we consider those with computing capability comparable to Jetson TX1. In particular, 11 state-of-the-art single-board computers, including NVIDIA Jetson TX2 [37], UDOO X86 ULTRA [47], Intel Aero Compute Board [48], LattePanda Alpha [49], Up Squared [50], Raspberry Pi 4 [51], NVIDIA Jetson AGX Xavier [52], DJI Manifold [53], HiKey960 [54], Rock960 [55] and Jetson TX1 [46], are found and compared in detail from various aspects (see Table 2.1 for the comparison results).

As shown in Table 2.1, Jetson AGX Xavier has the highest computing power and excels in memory capacity. However, it is the most power-consuming and costly, and it does not natively support Wi-Fi communications. Jetson TX2 with 6 CPU cores, 256-core NVIDIA Pascal GPU, and 8GB memory is the second powerful single-board computer. It provides an out-of-the-box high-throughput wireless local area network (WLAN) interface, and is also the smallest in size.

UDOO X86 ULTRA outperforms others in power consumption and operating system (OS) support; Intel Aero Compute Board is the lightest among those with known weight information; Raspberry Pi 4 is of the lowest cost; and Up Squared has the largest storage. All these single-board computers support virtualization.

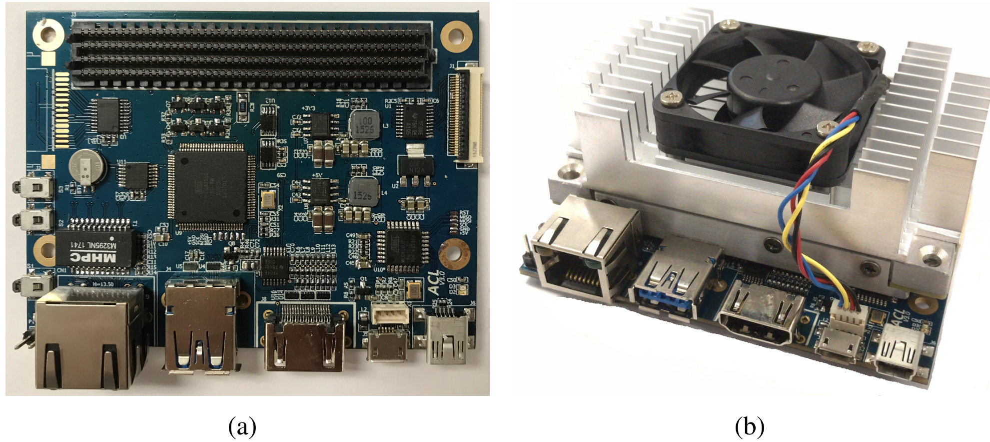
The above analysis provides us with guidelines to select proper single-board computers for UAV high-performance onboard computing. A trade-off should be achieved among different performance aspects based on the needs of specific applications. For instance, if flight time is more critical than real-time processing, UDOO X86 ULTRA that consumes less power or the lightweight Intel Aero Compute Board may be selected. If cost is of the major concern, Raspberry Pi 4 can be a good choice.

In this study, we select the NVIDIA Jetson TX2 as the computing hardware for the airborne computing platform. As shown in Table 2.1, its overall computing capability is above the average, especially considering the availability of a powerful GPU. Both the power consumption (7.5w) and weight (85g) are around the average. Another attractive factor is that there is an open-access online support community including FAQ and forum [56], which is very helpful for project developers.

While Jetson TX2 has a small size of 50mm$\times$87mm, the development board provided by NVIDIA is very large (17cm $\times$ 17cm). To address this issue, we design a new carrier board, as shown in Figure 2.1(a), based on the following technical specifications. The carrier board has a dimension of 88mm$\times$65mm with weight of 53g. The interfaces provided by the board include 1 HDMI, 3 UART, 1 CAN bus, 1 micro USB, 1 USB 2.0/3.0, 1 Ethernet port, 4 GPIO, and 2 camera ports. The carrier board with Jetson TX2 is shown in Figure 2.1(b).

2.2.3 A Prototype

With Jetson TX2 chosen as the computing unit, we then develop a prototype of the airborne computing platform [57, 9] that also incorporates a *quadcopter unit* for lifting and mobility, a *communication unit* for UAV-to-UAV (U2U), UAV-to-ground (U2G) and ground-to-



(a) (b)

Figure 2.1. a) A new Jetson TX2 carrier board and b) Jetson TX2 carrier board with the processor.

UAV (G2U) communications, and a *control unit* for addressing communications, networking, UAV navigation and application needs (see Figure 2.2).

Quadcopter Unit

The quadcopter unit serves as a platform carrier to carry other units. Compared with fixed-wing UAV, quadcopters are easier to operate, allow vertical taking off and landing, and can hover in the air. Here we select DJI Matrice 100 [58] as the quadcopter unit, due to its nice properties in terms of payload, expandability, stability and operability. For instance, the maximum weight allowed for the DJI Matrice 100 while taking off is 3.6kg, which exceeds the total weight of the whole system of 3.13kg. With a LiPo 6s battery, our prototype can fly for around 18 minutes.

Communication Unit

The communication unit supports the U2U and U2G/G2U communications. For U2U communication, we choose Ubiquiti Nanostation Loco M5 [59], a directional antenna, to enable long-range and broadband communication between two UAV. The transmission rate achieves up to 150 Mbps, allowing real-time video transmission. The maximum transmission distance is

10 km. For U2G/G2U communication, Huawei WS323 [60] is selected as the Wi-Fi router to enable communication between ground devices and the UAV, where ground devices connect to the router through the wireless local area network (WLAN). Through this link, sensor data such as videos captured by the UAV can be transmitted to the ground for visualization and analysis.

Control Unit

The control unit consists of two sub-units: UAV flight control and directional antenna control. In particular, the UAV flight control sub-unit makes the UAV follow desired trajectories, while maintaining stability. It translates high-level control commands received from the remote pilot to motor pulse width modulation (PWM) signals, based on UAV state measurements captured by sensors such as GPS and inertial measurement unit (IMU). The directional antenna control sub-unit controls the heading direction of the directional antenna to maximize the performance of directional communication. This sub-unit is composed of a rotating motor, a tunable plate, a motor driver and a compass. The tunable plate carries the directional antenna and the compass. To rotate the plate to a specified angle, the motor driver takes the control signal

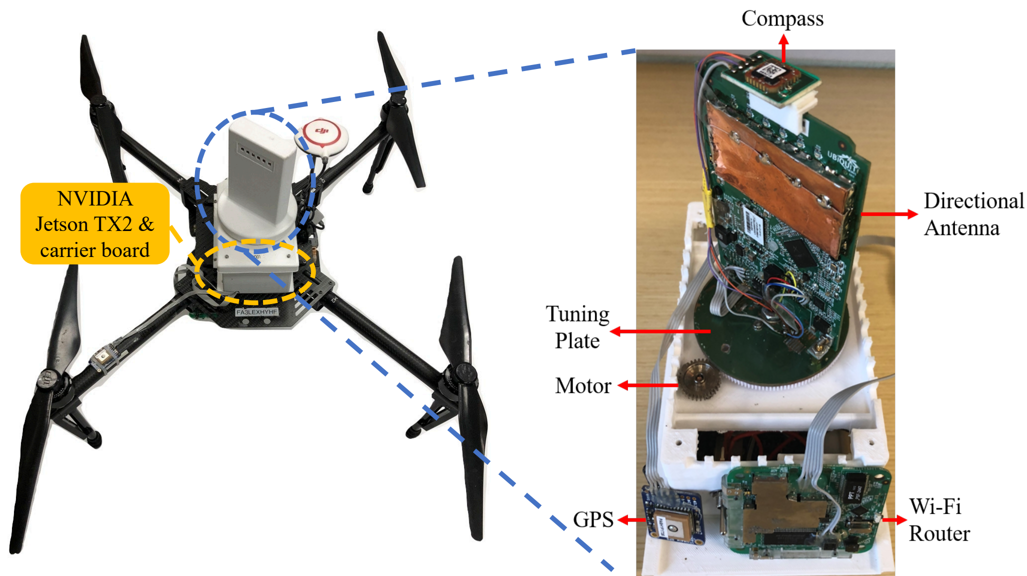


Figure 2.2. A prototype of the airborne computing platform.

generated by the computing unit and translates it to PWM signals. The motor then drives the plate to rotate based on the PWM signals. Here we select MTI-3-8A7G6T Xsens and Adafruit TB6612 as the compass and the motor driver, respectively.

2.3 Software Design for the Airborne Computing Platform

In this section, we investigate two key virtualization techniques, VM [61] using KVM and container using Docker [39], to improve the flexibility and programmability of the airborne computing platform. In particular, we first provide a brief overview of the current research status in the field of virtualization. To understand the impact of virtualization, we then conduct a series of experiments to study their performances from multiple aspects, including computing, networking, isolation, power consumption, etc., as well as the trade-offs among them.

2.3.1 Background and Related Work

To support diverse computing tasks on the same UAV platform, one of the key technologies is virtualization. First, virtualization provides powerful resource management capabilities, so that it can efficiently enable an application with specific computing requirements, such as CPU, memory, storage, networking, etc. Second, virtualization can facilitate concurrent execution of multiple applications on the same UAV. Third, in terms of security, virtualization can isolate unreliable and untrustworthy functionality, and improve the resilience of UAV to malicious attacks [62]. In addition to these advantages on a single UAV, virtualization can help to exploit the distributed computing capabilities on multiple connected UAV, which can evolve towards future generation of the networked airborne computing.

Virtualization has been studied extensively in the literature. The server-based virtualization has been mature and widely implemented in computing systems, especially the cloud [63, 64, 65]. Virtualization for mobile devices is more relevant to this study, which has aroused increasing attention with the wide use of mobile devices and the fast evolution of ARM processors, but is still under development [66]. In the last few years, several studies have investigated

the performance of virtualization on different mobile devices, such as Raspberry PI 2 [67], Cubieboard2 [68], ARM Chromebook [69], Banana Pi [70], and Insignal Arndale board [71], etc. However, since these studies were not directed to UAV, their performance analysis was limited to CPU, memory and disk in a single device. The unique features of UAV such as small payload, power constraint and real-time computing need, as well as the special characteristics of multi-UAV applications such as U2U communications and network connectivity were also not considered in these studies.

Virtualization has also played a critical role in emerging computing paradigms including IoT, fog computing and MEC. For instance, container-based virtualization is applied in [72, 73] to enable data processing at IoT devices. A Docker-based fog computing framework over Raspberry PI is described in [74]. In [75], the integration of IoT and fog computing with virtualization deployed in fog nodes is studied.

Despite the abundant works on virtualization, virtualization for UAV has been rarely studied. Among the limited studies we can find, paper [62] utilizes virtualization to enhance the resilience of UAV to malicious attacks, where Raspberry PI 2 is adopted as the onboard computing unit. Nutanix recently released a commercial UAV cloud platform, called Acropolis [76], which can hold multiple virtual machines (VMs). In [77], virtualization is implemented on fog servers to provide computing services for UAV fire detection. Overall, a comprehensive investigation of virtualization for UAV to enable high-performance onboard computing and advanced UAV applications is still lacking.

Virtualization can extend the computing capabilities of UAV, at a cost of performance overheads, due to resource partition, isolation and emulation. To understand the impact of virtualization, we next investigate the performances of two key virtualization techniques, KVM [38] and Docker [39], which are representatives of the hypervisor-based and container-based virtualization techniques, respectively. Please refer to [78] for a brief introduction of the two virtualization techniques, and the instructions to implement these techniques on Jetson TX2. In this study, both guest (VM or container) and host systems in Jetson TX2 implement Ubuntu

16.04 LTS with Linux kernel version 4.4 as the OS.

2.3.2 Computing Performance

In this subsection, we investigate the impact of KVM and Docker on the CPU and GPU computing performances of the proposed airborne computing platform, which is crucial for the success of many time-critical UAV applications.

Experimental Setup

To measure the computing performance of the airborne computing platform with virtualization capability, we create a VM that virtualizes CPU or GPU resources using KVM (or container using Docker) on the platform, and install the Rodinia Benchmark Suite [79] in the VM (or container). We then run the Stream Cluster (SC) application in the benchmark, which performs clustering for data streams. The execution time of the SC application indicates the computing performance. To reduce experimental uncertainty, each experiment is repeated for 10 times and the average execution time of the SC application is presented. This procedure is also applied to each experiment conducted in following studies.

In the experiment on the CPU performance, as the benchmark supports CPU multi-threading, which allows applications to be executed by multiple CPU cores in parallel, we vary the number of threads to test the parallel CPU computing performance. Note that each thread uses one CPU core. As only 4 ARM A57 CPU cores can be virtualized using KVM, up to 4 threads are evaluated for KVM. Docker successfully virtualizes all 6 CPU cores in Jetson TX2, and thus up to 6 threads are evaluated for Docker.

In the experiment on the GPU performance, as KVM does not support CUDA based GPU [80], we only evaluate the impact of Docker on the GPU performance, which adopts the PCI pass-through technique [81] to achieve GPU virtualization. We also compare the GPU performance of the airborne computing platform with its best CPU performance, i.e., using 6 threads. In both experiments, we vary the size of the input data stream in the SC application to

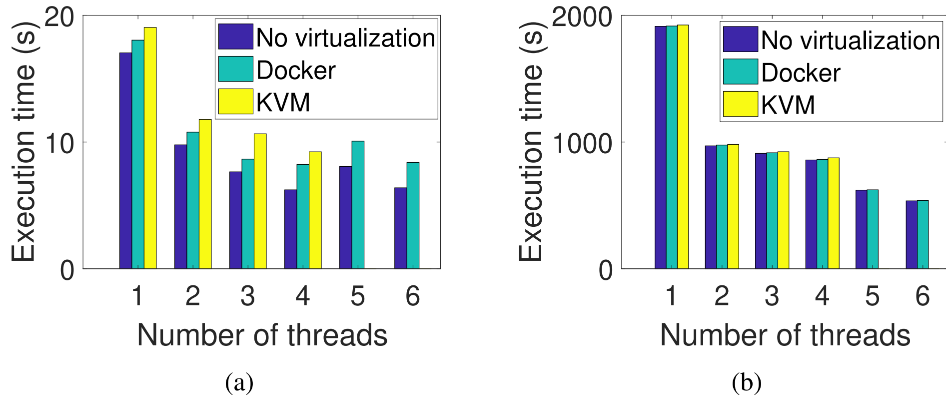


Figure 2.3. Execution time of the SC application implemented on CPU using increasing number of threads. The number of input points in the SC application is set to a) 10000 and b) 900000.

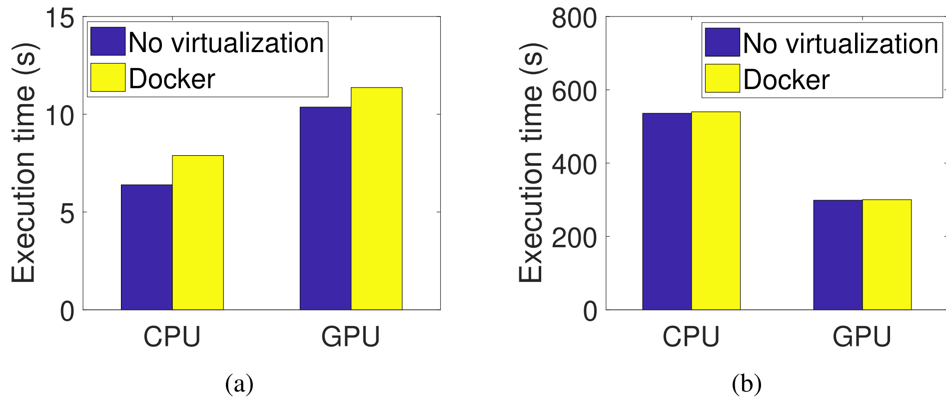


Figure 2.4. Execution time of the SC application implemented on GPU and CPU of six threads. The number of input points in the SC application is set to a) 10000 and b) 900000.

test the scalability of the computing platform.

Experimental Results

Figure 2.3 shows the CPU performance of the airborne computing platform before and after implementing KVM or Docker. As shown in the figure, both KVM and Docker introduce performance overheads, and KVM degrades the computing performance more. This is because KVM adopts more complicated procedures to allocate memory resources, in particular using second level address translation [38], while Docker achieves this by directly utilizing the Linux

system utility, i.e., control groups (cgroups) [39]. Our experiments also show that running 5 or 6 threads on Denver CPU cores does not improve the efficiency when the size of the data stream is small (see Figure 2.3(a)). This is due to the overheads for coordinating different CPU processors.

Figure 2.4 compares the performance of GPU and that of the CPU in two scenarios, i.e., before and after implementing Docker. As we can see from the figure, the computing performance of GPU is slightly worse than that of the hex-core CPU when the problem size is small, but GPU significantly outperforms the CPU when the problem size is large.

2.3.3 Networking Performance

In this subsection, we investigate the impact of KVM and Docker on the networking performance of the airborne computing platform, which is crucial for reliable and timely information sharing between UAV and ground mobile devices as well as among UAVs.

Experimental Setup

In this study, we conduct experiments to evaluate the networking performance for both the G2U/U2G and U2U communication links. To test the G2U/U2G communication link, we connect the airborne computing platform to a ThinkPad E540 laptop with the bandwidth of the Wi-Fi route set to 40 Mbps. To test the U2U communication link, we consider two scenarios: (1) the omni-directional antenna based short-distance communications and (2) directional antenna based long-distance communications. These two scenarios are tested by linking two computing platforms using the Wi-Fi router and the Ubiquiti Nanostation Loco M5, respectively.

To measure the networking performance, we install the Iperf benchmark [82] on airborne computing platforms and the Thinkpad laptop. This benchmark measures the throughput between two connected devices by sending data streams from one device (called client) to the other (called server). To obtain a comprehensive understanding of the networking performance, we vary the role of the airborne computing platform (client or server) and also vary the transmission protocol (TCP or UDP).

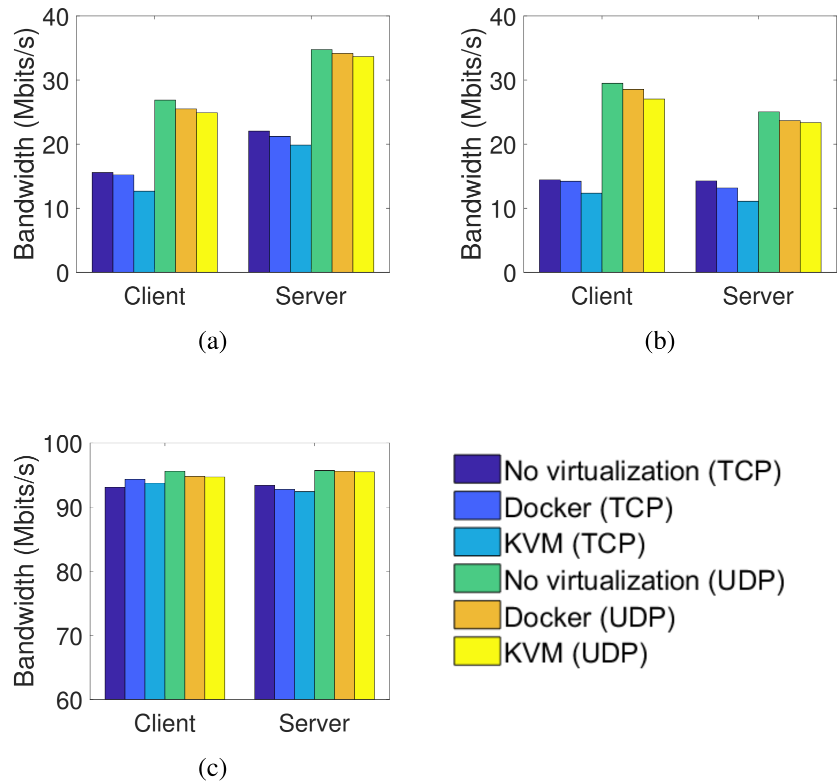


Figure 2.5. Bandwidth of the communication link a) between airborne computing platform and ThinkPad laptop, b) between two airborne computing platforms, and c) between two airborne computing platforms.

Experimental Results

The networking performance of the airborne computing platform before and after implementing KVM or Docker under different networking configurations is shown in Figure 2.5. In all these experiments, Docker shows less impact on the networking performance than KVM. This is due to the simplicity of Docker in network virtualization. In particular, unlike KVM that requires emulation of network devices in VMs to enable communications, Docker containers can directly build network connections by using the Linux system utilities, e.g., network namespace [39]. Another phenomenon observed in all experiments is that higher bandwidths are achieved when the UDP transmission protocol is adopted, as UDP sends packets continuously without acknowledgements.

Now let us analyze each subfigure. Figure 2.5(a) shows that the bandwidth of the G2U/U2G communications increases when the airborne computing platform acts as a server. This is mainly caused by the use of different network devices in Jetson TX2 and ThinkPad laptop. In cases when two identical airborne computing platforms are connected to simulate the U2U communications, the bandwidth measured at the server side is smaller than that measured at the client side (see Figures 2.5(b) & 2.5(c)). This is because VM or container requires port forwarding to receive packets, which introduces some overheads [83, 84]. The comparison between Figure 2.5(c) and the other two subfigures suggests that the high bandwidth is achieved by directional antennas, demonstrating their advantage over omni-directional antennas. Of interest, in Figure 2.5(c), when the TCP transmission protocol is adopted, the bandwidth measured at the client side increases after virtualization. This may be caused by the bridge network in KVM and Docker, which buffers packets sent from the guest to the host network interface and in turn helps alleviate traffic congestion and increases the packet transmission rate.

2.3.4 Isolation Performance

Virtualization can enhance the security of UAV applications, by isolating unreliable functionality. It can also enable concurrent execution of programs with different system requirements on the same airborne computing platform, by running these programs in different VMs (or containers). The success of these applications relies on how well VMs (or containers) are isolated, which is investigated in this subsection.

Experimental Setup

To test the isolation performance of KVM and Docker, we follow similar experimental setups in [85]. In particular, we create two guests (VMs or containers) in the airborne computing platform, and assign each guest with two ARM A57 CPU cores exclusively. We then install the Isolation Benchmark Suite (IBS) [86] to evaluate the isolation performance, which works by measuring the impact of a misbehaved guest (runs a stress test) on a well-behaved one (runs a

Table 2.2. Performance degradation of the well-behaved guest in different stress tests

	Docker	KVM
CPU	0.36%	0.41%
Memory	5.03%	6.0%
Disk I/O	2.56%	2.9%
Fork bomb	6.24%	1.28%
Network receiver	2.25%	4.68%
Network sender	1.73%	2.53%

baseline application). The smaller the impact is, the better the two guests are isolated. In this study, we run the Lower-Upper Gauss-Seidel solver (LU) application in the well-behaved guest, which performs a synthetic computational fluid dynamics calculation for a cubic region [87]. We here set the cubic size to $64 \times 64 \times 64$. In the misbehaved guest, we run different stress tests available in IBS to test the performance of KVM and Docker in isolating different hardware resources.

To evaluate the impact of the misbehaved guest on the well-behaved one, we measure the performance degradation of the well-behaved guest using the following equation:

$$\frac{T_s - T_n}{T_n} \times 100\% \tag{2.1}$$

where T_n and T_s represent the execution time of the LU application before and after running the stress test in the misbehaved guest, respectively.

Experimental Results

Table 2.2 shows the isolation performance of KVM and Docker in different stress tests. In particular, Docker demonstrates less performance degradation than KVM in the CPU, memory, disk I/O and network intensive stress tests, indicating relatively better performance in isolating these hardware resources. This is because Docker directly uses Linux namespaces to achieve

isolation, but KVM utilizes the hypervisor's trap-and-emulate mechanism that hangs the rest of the VMs when one traps to the hypervisor [38]. In the fork bomb test that generates large amount of processes to overwhelm the OS, the performance of Docker degrades significantly compared to KVM, as containers share the same OS kernel with the host system. Overall, both KVM and Docker perform well in isolating CPU resources, as different VMs or containers occupy different CPU cores. However, both are relatively weak in isolating the other hardware resources.

2.3.5 Power Consumption

In this section, we study the impact of KVM and Docker on power consumption, which directly influences the flight endurance of the proposed airborne computing platform.

Experimental Setup

To measure the power consumption of the airborne computing platform, we use the built-in 3-channel INA 3221 monitors in Jetson TX2 [88]. Two experiments are then conducted to evaluate the impact of KVM and Docker on the power consumption of the airborne computing platform at different operating conditions. Particularly, in the first experiment, the airborne computing platform does not run any applications, and its power consumption is measured before and after implementing KVM or Docker. In the second experiment, the power consumption of the platform when running the SC application is measured.

Experimental Results

The power consumption of the airborne computing platform before and after implementing KVM or Docker in the two experiments is shown in Figure 2.6. As we can see, both KVM and Docker increase the power consumption slightly in the two experiments. KVM consumes more power than Docker, as it introduces more overhead. Also note that compared with the power consumed by running the SC application, the power consumed by virtualization is negligible.

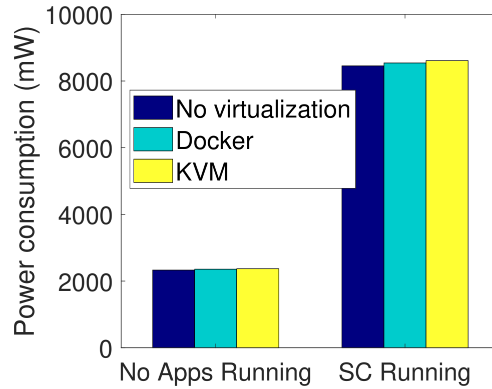


Figure 2.6. Power consumption of the airborne computing platform before and after running the SC application.

2.3.6 Discussions

In the above comparative studies, we evaluate the performances of KVM and Docker from the aspects of computing, networking, isolation and power consumption that are of major concern to UAV applications. In this subsection, we briefly discuss other performance aspects that are also of interest.

Resource Usage

Compared with VMs, containers consume fewer resources and thus can be quickly deployed. To demonstrate this feature, we conduct a simple experiment to measure the resource usage of a bare VM (or container) created by KVM (or Docker). No applications run in the VM or container. Table 2.3 summarizes the CPU, memory and storage usage of a bare VM or container measured using the *sysstat* and *free* Linux commands [89].

Table 2.3. Resource usage of a bare VM or container

	VM	**Container**
CPU	2.7%	0%
Memory	476 MB	0.3 MB
Storage	1.3 GB	103 MB

Live Migration

The live migration allows a running VM (or container) to be migrated from one computing platform to another, without interruptions during the migration process. Both Docker and KVM support live migration on server-based devices [90, 61]. However, live migration on mobile devices has been rarely studied and KVM currently does not support live migration on ARM-based devices. In addition, no solution is currently available for the Docker container-based live migration on Jetson TX2. We expect that this can be realized using checkpoints and restore utility [91], which we will leave to the future work.

OS Support

On ARM-based Linux single-board computers, KVM supports unmodified guest OSs [38], such as Ubuntu and openSUSE. However, Docker only supports ARM-based images [92], such as arm64v8/ubuntu, windows/nanoserver, and windows/iotcore.

Security

KVM is more resilient to malicious attacks than Docker. As VMs have their own OSs and kernels, the collapse of one VM does not influence others. However, containers share the kernel with the host OS. Therefore, once one container is attacked, other containers or even the whole system may collapse. For verification, we conduct a simple test. Specifically, we run the fork bomb, a denial-of-service attack, in the VM and container, respectively. As shown in Figure 2.7, the host OS works properly when the VM is under attack, but collapses when the container is attacked.

In summary, Docker achieves better performance than KVM in most aspects relevant to UAV applications, including computing, networking, isolation of CPU, memory, disk I/O and network resources, power consumption, and resource usage. Docker successfully virtualizes all CPU cores and GPU in Jetson TX2, but KVM can only virtualize the four ARM A57 CPU cores. On the other hand, KVM provides higher security. To enable more advanced UAV applications,

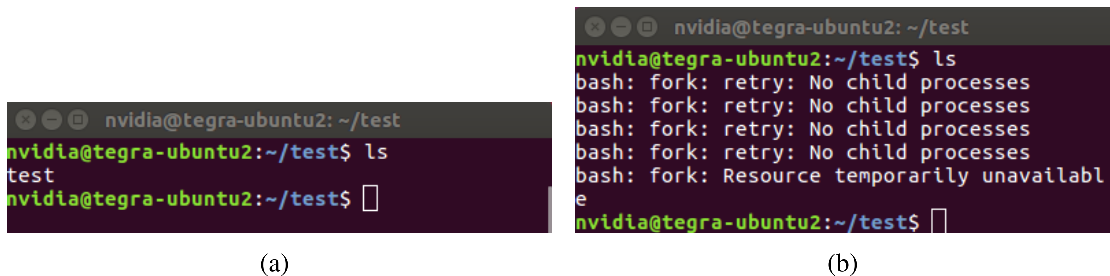


Figure 2.7. Response of the host OS when the a) VM or b) container is attacked.

the strengths of KVM and Docker need to be integrated. Live migration on ARM-based devices also needs to be realized for both KVM and Docker.

2.4 Performance of the Airborne Computing Platform

In this section, we investigate the performance of the proposed airborne computing platform in supporting real UAV applications. In particular, we first use OpenDroneMap for UAV image processing to illustrate the benefits of virtualization. We then further investigate the performance of two advanced UAV onboard computing tasks, real-time object detection and coded distributed computing.

2.4.1 OpenDroneMap

OpenDroneMap is an open-source UAV image processing software [93]. In this study, we first use the image resizing and 3-D model reconstruction functions in OpenDroneMap that require different amount of computing resources to investigate the impact of virtualization on the computing performance. Figure 2.8 shows the average execution time of the two functions to process a UAV image with size of 3.9MB in different virtualization environments[1]. The results demonstrate the advantage of Docker over KVM, especially in computing complicated UAV onboard computing tasks. The 3-D geographical model reconstructed from 41 2-D UAV images is shown in Figure 2.9.

[1]UAV images are downloaded through this link: https://github.com/OpenDroneMap/odm_data_copr.git.

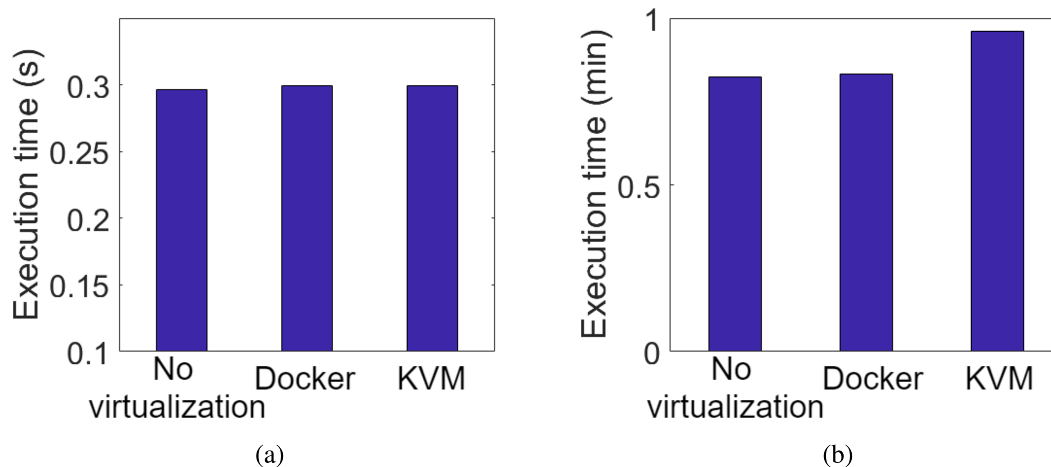


Figure 2.8. Execution time of the a) image resizing and b) 3-D model reconstruction functions in OpenDroneMap to process a UAV image in different virtualization environments.

We next use OpenDroneMap to demonstrate the benefits of virtualization in facilitating resource management for UAV. Generally, virtualization provides developers with the convenience to run multiple applications simultaneously without considering resource sharing and context switches among processes, which will increase the execution time of the applications. To illustrate this fact, we first conduct experiments to show the consequence of running two applications simultaneously when virtualization is not applied. Figure 2.10(a) shows the execution time of the LU application (cubic size is set to $36 \times 36 \times 36$) and the image resizing function in OpenDroneMap (processes 41 UAV images) when they run separately on the airborne computing platform compared to the case when they run simultaneously. As we can see from this figure, when the two applications run simultaneously, the execution time of both applications increases significantly, and the LU application even takes more time than the OpenDroneMap application.

We then implement virtualization on the airborne computing platform, and run the two applications simultaneously but in different guests. For better overall performance, the guest that runs the LU application is allocated with 1 CPU core and the one that runs the more computation-intensive OpenDroneMap application is allocated with 3 CPU cores. As shown in Figure 2.10(b), virtualization helps improve the overall computing performance significantly through resource

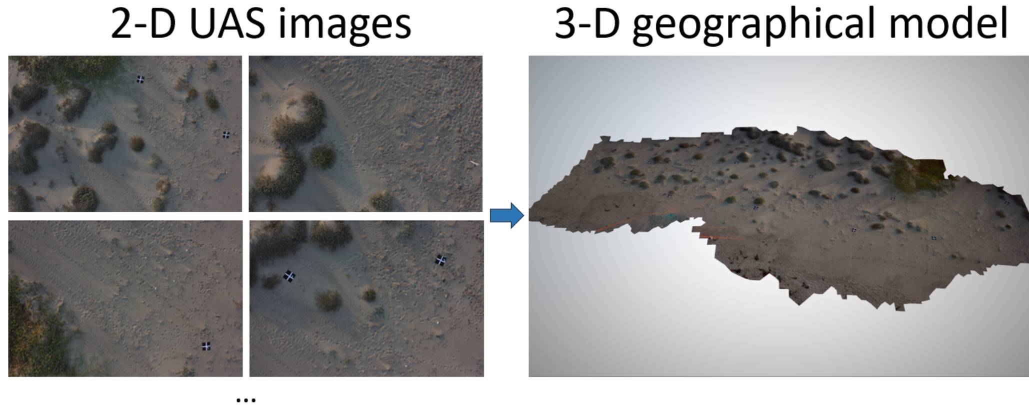


Figure 2.9. The 3-D geographical model generated from 41 UAV images using the 3-D model reconstruction function in OpenDroneMap.

allocation and isolation, despite the associated overhead.

2.4.2 Real-Time Object Detection

Real-time object detection is crucial for many UAV applications including search and rescue, traffic monitoring, infrastructure inspection, and reconnaissance. As this type of task is computationally demanding, it is typically executed at ground stations or the cloud. In this study, we show that real-time object detection can be achieved onboard of UAV even with virtualization.

Consider the scenario where UAV is dispatched to detect and track humans in a search and rescue mission. To achieve this, we implement a deep neural network (DNN) model [94] on the airborne computing platform, which is built on GPU and uses NVIDIA TensorRT and cuDNN. This model has been pre-trained for human detection. We use 10 images captured from a UAV action video [95] to evaluate the average execution time of this model to process a UAV image. In particular, the average recognition times of the airborne computing platform without virtualization and with Docker container-based virtualization are around 0.129s and 0.148s per image of size 850KB, respectively. This demonstrates the feasibility of performing real-time object detection onboard of UAV even with virtualization. Note that it takes around 0.253s and 0.267s to transmit a single image of size 850KB from the airborne computing platform without

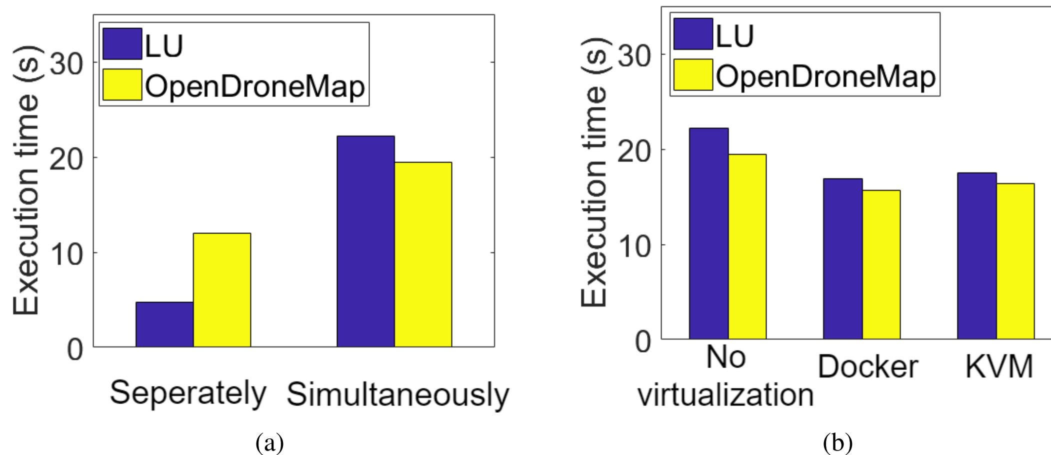
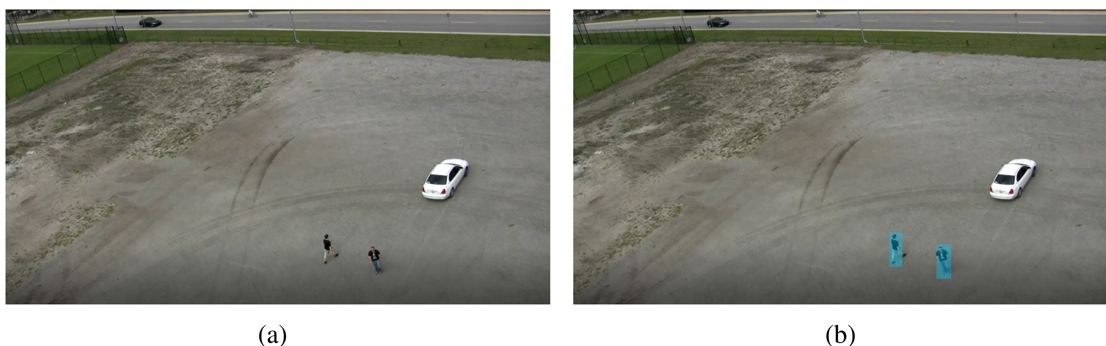


Figure 2.10. Execution time of two applications a) when running separately and when running simultaneously in Jetson TX2 without virtualization, b) when running simultaneously in Jetson TX2 of three different virtualization setups.

virtualization and with Docker-based virtualization, respectively, to the ground (Thinkpad laptop) through omni-directional antenna- and UDP-based communication, according to the results shown in Figure 2.5(a). Figure 2.11 illustrates the accuracy of the DNN model in recognizing humans on a UAV image.



(a) (b)

Figure 2.11. A UAV image a) before and b) after applying the DNN based object detection.

2.5 Conclusion

In this chapter, we developed a new UAV-based airborne computing platform to address the onboard computing limitations of existing UAV platforms so as to support UAV-enabled MEC and to enable more advanced UAV applications. This airborne computing platform was designed from three aspects: hardware, software and applications. To design the hardware, we first investigated the desired features for the onboard computing hardware, and then conducted a comprehensive comparison study among state-of-the-art single-board computers to select a suitable one as the computing unit. A prototype was then designed and implemented, which not only contains the computing unit, but also hardware for UAV mobility, communications and control. To design the software, we investigated two representative virtualization techniques, VM using KVM and container using Docker, and evaluated their performances from various aspects. Through comprehensive experimental studies, we find that Docker outperforms KVM in most performance aspects, including computing, networking, isolation of most hardware resources, power consumption, and resource usage. Docker also successfully virtualizes all CPU cores and GPU in Jetson TX2. On the other hand, KVM is more secure. Finally, we studied three real UAV applications, including UAV image processing, real-time object detection, and coded distributed computing, to demonstrate the performance, applicability and potentials of the proposed airborne computing platform.

2.6 Acknowledgement

This Chapter, in part, is a reprint of the published journal: B. Wang, J. Xie, S. Li, Y. Wan, Y. Gu, S. Fu, K. Lu, “Computing in the Air: An Open Airborne Computing Platform”, *IET Communications*, Vol.14, pp. 2410-2419, 2020.

Chapter 3

Batch-Processing Based Coded Computing for Static Networked Airborne Computing Systems

3.1 Introduction

As the airborne computing capacity of individual UAVs is still limited due to the small payload, this chapter aims to further enhance their computing capacity by leveraging the computing resources of neighboring UAVs through networking and computing resource sharing. This can be achieved by using distributed computing techniques.

Distributed computing has been widely adopted to perform various computation tasks in different computing systems [96, 97, 98]. For instance, to perform big data analytics in cloud computing systems, MapReduce [99] and Apache Spark [100] are the two prevalent modern distributed computing frameworks that process data in the order of petabytes. Despite the importance of distributed computing, many design challenges remain. One major challenge is that many computing frameworks are vulnerable to uncertain disturbances, such as node/link failures, communication congestion, and slow-downs [101]. Such disturbances, which can be modeled as stragglers that are slow or even fail in returning results, have been observed in many large-scale computing systems such as cloud computing[102], mobile edge computing[103], and fog computing[104].

A variety of solutions have been developed in the literature to address stragglers. For example, the authors of [105] proposed to identify and blacklist nodes that are in bad health and to run tasks only on well-performed nodes. However, empirical studies show that stragglers can occur in non-blacklisted nodes as well [106, 107]. As another type of solution, delayed computation tasks can be re-executed in a speculative manner [99, 108, 105, 109]. Nevertheless, such speculative execution techniques have to wait to collect the performance statistics of the tasks before generating speculative copies and thus have limitations in dealing with small jobs [107]. To avoid waiting and predicting stragglers, the authors of [110, 107] suggested to execute multiple clones of each task and use results generated by the fastest clones. Although their results show the promising performance of this approach in reducing the average completion time of small jobs, the extra resources required for launching clones can be considerably large, because multiple clones are executed for each task.

Instead of directly replicating the whole task, the *coding* techniques can be adopted to introduce arbitrary redundancy into the computation in a systematic way. However, until a few years ago, the coding techniques have been mostly known for their capability in improving the resilience of communication, storage and cache systems to uncertain disturbances [111, 112, 113]. Lee *et al.* [114, 115] presented the first *coded distributed computing* (CDC) scheme to speed up matrix multiplication and data shuffling. Since then, CDC has attracted significant attention in the distributed computing community. To understand the key idea of CDC, consider the matrix multiplication problem that aims to multiply a large input matrix X with another pre-stored matrix A. To reduce the computation time, the traditional approach (see Figure 3.1(a) for an illustration) distributes the task by storing sub-matrices A_i of A at different computing nodes called worker nodes, where $A = [A_1; A_2; \cdots ; A_n]$ and n is the total number of sub-matrices. To compute AX, a master node first sends X to all worker nodes. Each worker node then computes A_iX and sends the result back to the master node. As the master node cannot recover AX until all results are received, the efficacy of this approach is bounded by the slowest worker node, i.e., the straggler. The coded computation (see Figure 3.1(b) for an illustration) addresses this issue by

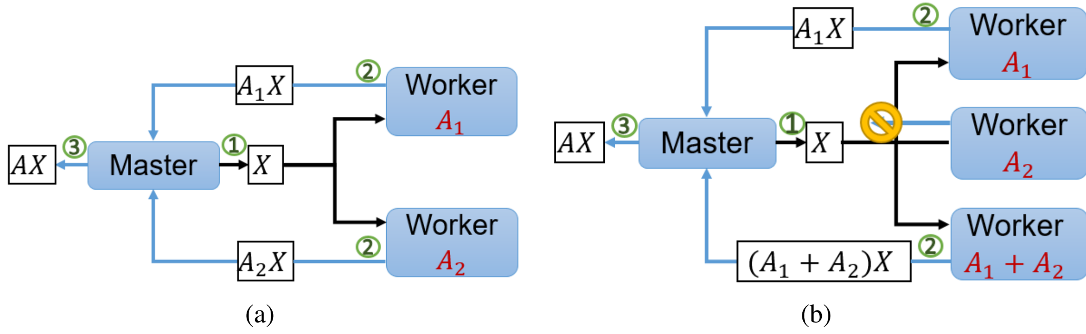


Figure 3.1. Illustration of the a) uncoded and b) coded computation to perform matrix multiplication with $n = 2$. The numbers marked in green describe the computation procedures.

introducing redundancy into the computation through erasure codes. For instance, consider the matrix multiplication problem with $n = 2$, the coded approach introduces an additional worker node that stores $A_1 + A_2$. Therefore, the master node can recover AX upon receiving the results from any two worker nodes. For instance, if A_1X and $(A_1 + A_2)X$ arrive at the master node first, AX can be recovered by $AX = [A_1X; (A_1 + A_2)X - A_1X]$.

To evaluate the performances of the uncoded and coded computations, we implement each approach on the airborne computing platform introduced in Chapter 2 to solve the matrix multiplication problem with $n = 2$. In particular, we create three containers as the worker nodes in one airborne computing platform, and one container as the master node in another platform. Each container is assigned with one CPU core. Containers on different airborne computing platforms are linked through the Loco M5 to simulate the directional-antenna based long-distance UAV-to-UAV communications. We then create two 800×800 random matrices, A and X. Matrix A is divided equally into two sub-matrices A_1 and A_2 of size 400×800. The execution time of the two approaches to compute AX is provided in Table 3.1, where two scenarios are evaluated. In the first scenario, only matrix multiplication is performed within each container and thus no straggler exists. In the second scenario, a CPU stress test [86] is executed concurrently in one of the worker nodes to consume its computing resources. This worker node thus becomes a straggler. The results shown in Table 3.1 illustrate the robustness of the coded computation to

Table 3.1. Execution time of uncoded and coded matrix multiplication with $n = 2$

	No straggler exists	**Straggler exists**
Uncoded Computation	13.09s	24.58s
Coded Computation	13.69s	13.91s

system noises such as stragglers.

Although a variety of CDC schemes have been developed to solve different computation problems, most of these schemes assume homogeneous computing nodes, which is not a common case in realistic scenarios. Moreover, they require each worker node to first complete the computation task and then send back the whole result to the master node, which introduces significant delays [24, 25, 115, 114].

In this chapter, we focus on the matrix-vector multiplication problem and propose a novel coding scheme, called batch-processing based coded computing (BPCC), to speed up the computational efficiency of general distributed computing systems with heterogeneous computing nodes and improve their robustness to uncertain disturbances. Unlike most existing CDC schemes, our BPCC allows each node to return partial computing results to the master node in batches before the whole computation task is completed. Therefore, BPCC achieves lower latency. Also worthy of note is that the partial results can be used to generate approximated solutions, e.g., by applying the singular value decomposition (SVD) approach in [116], which is very useful for applications that require timely but unnecessarily optimized decisions such as emergency response. To the best of our knowledge, such a BPCC framework has not been fully investigated in the literature.

The contributions of this chapter are summarized as follows:

1. *An optimal load allocation strategy.* For systems with heterogeneous computing nodes, equally distributing the computation load may lead to bad performance. To optimize the computational efficiency, we formulate an optimization problem for general BPCC with the assumption that the processing time of each computing node follows a shifted exponential

distribution. To solve the optimization problem, we formulate alternative optimization problems, based on which we design an optimal load allocation scheme that assigns proper amount of load to each node to achieve the minimal expected task completion time.

2. *Comprehensive theoretical analyses.* We conduct formal theoretical analyses to prove the asymptotic optimality of BPCC and the impact of its important parameter. We also prove that it outperforms a state-of-the-art CDC scheme for heterogeneous systems, called Heterogeneous Coded Matrix Multiplication (HCMM)[24, 25].

3. *Extensive simulation and real experimental studies.* To further demonstrate the performance of BPCC, we compare it with three benchmark schemes, including the Uniform Uncoded, Load-Balanced Uncoded, and HCMM. The simulation results show the impact of BPCC parameters including number of batches and number of worker nodes. Specifically, the efficiency of BPCC improves with the increase of the number of batches and the solution of BPCC is optimal when the number of worker nodes approaches infinity. A sensitivity study shows the performance of BPCC when parameters in the computing model take erroneous values. Moreover, the simulation results also demonstrate that BPCC can improve computing performance by reducing the latency up to 73%, 56%, and 34% over the aforementioned three benchmark schemes, respectively. In the real experiments, we test all distributed computing schemes in the Amazon EC2 computing clusters. In particular, we deploy a heterogeneous computing cluster that consists of different machine instances in Amazon EC2. The results show that our BPCC scheme is more efficient and robust to uncertain disturbances than the benchmark schemes.

3.2 Related Work

Following the seminal work in [112, 114, 115], many different computation problems have been explored using codes, such as the gradients [117], large matrix-matrix multiplication [118], linear inverse problems [119], and nonlinear operations [120]. Other relevant coded

computation solutions include the "Short-Dot" coding scheme [121] that offers computation speed-up by introducing additional sparsity to the coded matrices and the unified coded framework [122, 123] that achieves the trade-off between communication load and computation latency.

While most CDC schemes consider homogeneous computing nodes, there have been a few recent studies that investigated CDC over heterogeneous computing clusters. In particular, Kim *et al.* [124, 125] considered the matrix-vector multiplication problem and presented an optimal load allocation method that achieves a lower bound of the expected latency. Reisizadeh *et al.* [24] introduced a different approach, namely Heterogeneous Coded Matrix Multiplication (HCMM), that can maximize the expected computing results aggregated at the master node. In [24, 25], the authors proved that the HCMM is asymptotically optimal under the assumption that the processing time of each computing node follows a shifted exponential or Weibull distribution. Also of interest, Keshtkarjahromi *et al.* [126] considered the scenario when computing nodes have time-varying computing powers, and introduced a coded cooperative computation protocol that allocates tasks in a dynamic and adaptive manner. Narra *et al.* [127] also developed an adaptive load allocation scheme and utilized a LSTM-based model to predict the computation capability of the worker nodes.

To reduce the output delay, there have been some attempts to enable early return of partial results [116, 128, 129]. In particular, an anytime coding technique was introduced in [116], which adopts the SVD to allow early output of approximated result. Also of interest is the study presented in [128], which introduced a hierarchical approach to address the limitations of above coding techniques in terms of wastefully ignoring the work completed by slow worker nodes. In particular, to better utilize the work completed by each worker node, it partitions the total computation at each worker node into layers of sub-computations, with each layer encoding part of the job. It then processes each layer sequentially. The final result can be obtained after the master node recovers all layers. The simulation results demonstrate the effectiveness of this approach in reducing the computation latency. However, as the worker nodes have to process the

layers in the same order, the results obtained by slow worker nodes for layers that have already been recovered are useless. Furthermore, this approach, as well as aforementioned approaches, assumes homogeneous computing nodes. Another relevant study is presented in [129], which introduced a rateless fountain coding scheme that can utilize partial results returned by worker nodes.

3.3 System Models

In this section, we first introduce the computing system for distributed matrix-vector multiplication. We then illustrate three computing schemes, including the proposed batch processing-based coded computing (BPCC). Finally, we formulate an optimization problem for BPCC.

3.3.1 Computing System

We consider a distributed computing system that consists of one master node and N ($N \in \mathbb{Z}^+$) computing nodes, a.k.a., worker nodes. Using this system, we investigate how to quickly solve a matrix-vector multiplication problem, which is one of the most basic building blocks of many computation tasks. Specifically, we consider a matrix-vector multiplication problem $y = Ax$, where $y \in \mathbb{R}^r$ is the output vector to be calculated, $x \in \mathbb{R}^m$ is the input vector to be distributed from a master node to multiple workers, and $A \in \mathbb{R}^{r \times m}$ is an $r \times m$ dimensional matrix pre-stored in the system. Both r and m can be very large, which implies that calculating Ax at a single computing node is not feasible. Finally, we define $[n] = \{1, 2, \dots, n\}$, where n is an arbitrary positive integer, i.e., $n \in \mathbb{Z}^+$.

3.3.2 Computing Schemes

Uncoded Distributed Computing

To solve the above problem, a traditional distributed computing scheme divides matrix A into a set of sub-matrices $A_1, A_2, \cdots, A_N$, and pre-stores each sub-matrix $A_i \in \mathbb{R}^{\ell_i \times m}$ in computing

node i, where $\forall i \in [N], \ell_i \in \mathbb{Z}^+$ and $\sum_{i=1}^{N} \ell_i = r$. Upon receiving the input vector x, the master node sends vector x to all worker nodes. Each worker node i then computes $y_i = A_i x$ and returns the result to the master node. After all results are received, the master node aggregates the results and outputs $y = [y_1^T, y_2^T, \cdots, y_N^T]^T$, where ^{T} stands for transpose.

Due to the existence of uncertain system disturbances, the uncoded computing scheme may defer or even fail the computation, because the delay or loss of any y_i, $i \in [N]$, will affect the calculation of the final result $y = Ax$. To address this issue, more computing nodes can be used to perform distributed computing. For instance, the master node can have two or more computing nodes to compute y_i. This approach, however, is not efficient because the cost can be unnecessarily large.

Coded Distributed Computing (CDC)

In recent years, a more efficient computing paradigm, CDC, has been introduced to tackle the issue of uncertain disturbances. There are many CDC schemes in the literature, and we consider a generic CDC scheme as follows.

In this CDC scheme, A will first be used to calculate a larger matrix $\hat{A} \in \mathbb{R}^{q \times m}$ with more rows, i.e., $q > r$, by using $\hat{A} = HA$, where $H \in \mathbb{R}^{q \times r}$ is the encoding matrix with the property that any r row vectors are linearly independent from each other [120]. In other words, we can use any r rows of H to create an $r \times r$ full-rank matrix. Note that this encoding procedure is performed offline and $\hat{A}$ can be considered to be pre-stored in the system. Similar to the uncoded computing scheme, matrix $\hat{A}$ can then be divided into N sub-matrices $\hat{A}_1, \hat{A}_2, \cdots, \hat{A}_N$, where $\hat{A}_i \in \mathbb{R}^{\ell_i \times m}, \forall i \in [N]$, $\sum_{i=1}^{N} \ell_i = q$, and each worker node i calculates $\hat{y}_i = \hat{A}_i x$.

Different from the uncoded computing scheme, the master node does not need to wait for all worker nodes to complete their calculations, because it can recover Ax once the total number of rows of the received results is equal to or larger than r. In particular, suppose the master node

receives $\hat{y}_b \in \mathbb{R}^r$ at a certain time t, it can first infer that $\hat{y}_b$ must satisfy

$$\hat{y}_b = \hat{H}_b A x,$$

where $\hat{H}_b \in \mathbb{R}^{r \times r}$ is a sub-matrix of the encoding matrix H corresponding to $\hat{y}_b$. The master node can then calculate

$$y = Ax = \hat{H}_b^{-1} \hat{y}_b. \tag{3.1}$$

BPCC

In the literature, most existing CDC schemes assume that each worker node i sends the complete $\hat{y}_i$ to the master node when it is ready, which may incur large delays. To further speed up the computation, we propose a novel BPCC scheme and the main idea is to allow each worker node to return *partial results* to the master node.

Specifically, we consider that each worker node i equally divides the pre-stored encoded matrix $\hat{A}_i$ row-wise into p_i sub-matrices, named as *batches*, where $p_i \in \mathbb{Z}^+$ is the number of batches and $p_i \leq \ell_i$. Except the last batch, each batch has $\lceil \frac{\ell_i}{p_i} \rceil = b_i$ rows. After receiving the input vector x from the master node, the worker node multiplies each batch with x and will send back the partial results once available. Suppose that the master node receives $s_i(t)$ batches from the worker node i by time t, where $0 \leq s_i(t) \leq p_i$, it can then recover the final result when $\sum_{i=1}^{N} \min(\ell_i, s_i(t) b_i) \geq r$, by using Eq. (3.1).

3.3.3 Problem Formulation

In the previous sub-section, we introduced the key idea of the BPCC scheme. In the following study, we focus on optimizing the performance of BPCC. Specifically, we consider minimizing the task completion time. This is achieved by allocating proper computation load (i.e., ℓ_i) to each worker node.

We first consider the behavior of waiting time, which is defined as the duration from the time that the master node distributes x to the time that it receives a certain result. For BPCC, we

let $T_{k,i}$ be the waiting time for the master node to receive k batches from worker node i, $k \in \mathbb{Z}^+$. Clearly, $T_{k,i}$ can be modeled as a random variable following a certain probability distribution. Following the modeling technique used in recent studies [25, 114, 115, 128], we consider that $T_{k,i}$ follows a shifted exponential distribution defined below:

$$\Pr(T_{k,i} \leq t) = \begin{cases} 1 - e^{-\mu_i(\frac{t}{kb_i} - \alpha_i)} & \text{if } t \geq kb_i\alpha_i \\ 0 & \text{otherwise,} \end{cases} \tag{3.2}$$

where μ_i and α_i are straggling and shift parameters, respectively, and μ_i and α_i are **positive** constants for all $i \in [N]$. Furthermore, we assume that $T_{k,i}$ is independent from $T_{k',j}$, $\forall j \in [N]$, $j \neq i$, $k' \in \mathbb{Z}^+$.

To facilitate further analysis, we assume that the computation task scales with N, i.e., $r = \Theta(N)$. Next, we assume that the computing nodes are fixed with time-invariant computation capabilities, and the network maintains a stable communication delay during the computing process.

We now define T as the amount of time to complete a computation task. Based on the above definitions and assumptions, we see that T must satisfy $\sum_{i=1}^{N} s_i(T)b_i \geq r$. Given the number of batches for each worker node $p = (p_1, p_2, \ldots, p_N)$, where $p_i \in \mathbb{Z}^+$, $\forall i \in [N]$, the optimization can be formulated as follows:

$$\begin{aligned} \mathscr{P}_{\text{main}} : \underset{\ell}{\text{minimize}} \quad & \mathbb{E}[T] \\ \text{subject to} \quad & \ell_i \in \mathbb{Z}^+, \forall i \in [N] \\ & \ell_i \geq p_i, \forall i \in [N] \end{aligned} \tag{3.3}$$

where $\ell = (\ell_1, \ell_2, \ldots, \ell_N)$.

In the following sections, we will first discuss how to solve the optimization problem, in which we will conduct theoretical analysis to show the optimality and advantages of BPCC. We

will then conduct extensive simulation and real experimental studies to validate the assumptions and to evaluate performance of the optimization algorithm.

3.4 Main Results

In this section, we aim to solve the optimization problem $\mathscr{P}_{\text{main}}$. In particular, we will first provide a simplified formulation, for which we then apply a two-step alternative formulation. Next, we show how to solve the alternative problems and prove the optimality of the solution. We then analyze the impact of parameter $p_i, \forall i \in [N]$ on the solution, and finally prove that this solution outperforms a recent CDC scheme without batch processing.

3.4.1 Notations for Asymptotic Analysis

For any two given functions $f(n)$ and $g(n)$, $f(n) = \Theta(g(n))$ if and only if there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$; $f(n) = \mathscr{O}(g(n))$ if and only if there exist constants n_0 and c such that $f(n) \leq c g(n)$ for all $n \geq n_0$; and $f(n) = o(g(n))$, if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

3.4.2 A Simplified Formulation

We relax the constraint from $\ell_i \in \mathbb{Z}^+$ to $\ell_i \geq 0$, $\forall i \in [N]$ to simplify the analysis. We also remove the constraint $\ell_i \geq p_i$, $\forall i \in [N]$, by assuming that $p_i \in \mathbb{Z}^+$ is properly selected such that the optimal solution satisfies this constraint. Consequently, the problem in Eq. (3.3) can be formulated as follows:

$$\begin{aligned} \mathscr{P'}_{\text{main}} : \underset{\ell}{\text{minimize}} \quad & \mathbb{E}[T] \\ \text{subject to} \quad & \ell_i \geq 0, \forall i \in [N], \end{aligned}$$

Once the above problem is solved, we can round each optimal load number ℓ_i up to its nearest integer using the ceiling function (denoted as $\lceil \; \rceil$). Note that the effect of this rounding step is negligible in practical applications with large load numbers, such as those considered in our simulation and experimental studies[25]. In cases when the derived load number ℓ_i is smaller

than p_i, we reduce the value of p_i until this assumption holds. Note that we can always find such p_i that satisfies the constraint, as the derived load number ℓ_i is always larger than or equal to 1.

3.4.3 A Two-Step Alternative Formulation

To solve the above problem, which is NP-Hard, we provide a two-step alternative formulation, inspired by [25]. We will show later that this alternative formulation provides an asymptotically optimal solution to problem $\mathscr{P}'_{\text{main}}$.

The key idea of the two-step alternative formulation is to first maximize the amount of results accumulated at the master node by a feasible time t, i.e., $t \geq \max_i\{\alpha_i \ell_i\}$, and then minimize time t such that sufficient amount of results are available to recover the final result. In particular, we let $S(t) = \sum_{i=1}^{N} s_i(t) b_i$ be the amount of results received by the master node by time t, where $b_i = \frac{\ell_i}{p_i}$ is the batch size. For a feasible time t, we first maximize the expected amount of results received by the master node, through solving the following problem:

$$\begin{aligned} \mathscr{P}_{\text{alt}}^{(1)} : \underset{\ell}{\text{maximize}} \quad & \mathbb{E}\left[S(t)\right] \\ \text{subject to} \quad & \ell_i \geq 0, \forall i \in [N] \end{aligned}$$

After obtaining the solution to $\mathscr{P}_{\text{alt}}^{(1)}$, denoted as $\ell^*(t) = (\ell_1^*(t), \cdots, \ell_N^*(t))$, we then minimize the time t such that there is a high probability that the results received by the master node by time t are sufficient to recover the final result, by solving

$$\begin{aligned} \mathscr{P}_{\text{alt}}^{(2)} : \text{minimize} \quad & t \\ \text{subject to} \quad & \Pr\left[S^*(t) < r\right] = o\left(\frac{1}{N}\right) \end{aligned}$$

where $S^*(t)$ is the amount of results received by the master node by time t for load allocation $\ell^*(t)$.

3.4.4 Solution to the Two-Step Alternative Problem

To solve the two-step alternative problem, we first consider $\mathscr{P}_{\text{alt}}^{(1)}$. Note that, the expected amount of results received by the master node by time t is:

$$\begin{aligned}\mathbb{E}[S(t)] &= \sum_{i=1}^{N} \mathbb{E}[s_i(t) b_i] \\ &= \sum_{i=1}^{N} b_i \left[\sum_{k=1}^{p_i} k \Pr[s_i(t) = k] \right]\end{aligned} \tag{3.4}$$

where $s_i(t)$ is an integer in range $0 \leq s_i(t) \leq p_i$, and $\Pr[s_i(t) = k]$ is the probability that the master node receives exactly k batches from worker node i,

$$\begin{aligned}&\Pr[s_i(t) = k] \\ &= \begin{cases} 1 - \Pr(T_{1,i} \leq t), & k = 0 \\ \Pr(T_{k,i} \leq t) - \Pr(T_{k+1,i} \leq t), & 0 < k < p_i \\ \Pr(T_{p_i,i} \leq t). & k = p_i \end{cases}\end{aligned}$$

$\mathbb{E}[S(t)]$ in Eq. (3.4) can then be computed by:

$$\begin{aligned}\mathbb{E}[S(t)] &= \sum_{i=1}^{N} b_i \left[\sum_{k=1}^{p_i - 1} k \Pr[s_i(t) = k] + p_i \Pr[s_i(t) = p_i] \right] \\ &= \sum_{i=1}^{N} \sum_{k=1}^{p_i} b_i \Pr(T_{k,i} \leq t) \\ &= \sum_{i=1}^{N} \sum_{k=1}^{p_i} b_i \left(1 - e^{-\mu_i (\frac{t}{k b_i} - \alpha_i)} \right) \\ &= \sum_{i=1}^{N} \left(\ell_i - b_i \sum_{k=1}^{p_i} e^{-\mu_i (\frac{t}{k b_i} - \alpha_i)} \right) \\ &= \sum_{i=1}^{N} \left(\ell_i - \frac{\ell_i}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i (\frac{t p_i}{k \ell_i} - \alpha_i)} \right)\end{aligned} \tag{3.5}$$

The solution to $\mathscr{P}_{\text{alt}}^{(1)}$ can then be obtained by solving the following equation for each $i \in [N]$:

$$\frac{\partial}{\partial \ell_i}\mathbb{E}\left[S(t)\right] = 1 - \left[\sum_{k=1}^{p_i}\left(\frac{1}{p_i} + \frac{\mu_i t}{\ell_i k}\right) e^{-\mu_i(\frac{t p_i}{k \ell_i} - \alpha_i)}\right] = 0,$$

which yields:

$$\ell_i^*(t) = \frac{t}{\lambda_i} \tag{3.6}$$

λ_i is the positive solution to the following equation:

$$\sum_{k=1}^{p_i}\left(\frac{1}{p_i} + \frac{\mu_i \lambda_i}{k}\right) e^{-\mu_i(\frac{\lambda_i p_i}{k} - \alpha_i)} = 1, \tag{3.7}$$

which is a constant independent of t. To show that Eq. (3.7) has a single positive solution, we can define an auxiliary function f_i for each i:

$$f_i(x) = \sum_{k=1}^{p_i}\left(\frac{1}{p_i} + \frac{\mu_i x}{k}\right) e^{-\mu_i(\frac{x p_i}{k} - \alpha_i)}.$$

We can see that $f_i(x)$ decreases monotonically with the increase of x when $x > 0$. We can also find that $f_i(0) = e^{\mu_i \alpha_i} > 1$ and $f_i(\infty) = 0$. Based on these statements, we know that a unique λ_i exists and can be efficiently solved using a numerical approach. Next, we show in Lemma 1 that λ_i has closed-form infimum and supremum.

Lemma 1. *Let λ_i, $i \in [N]$, be the positive solution to Eq. (3.7). Its infimum is given by*

$$\inf \lambda_i = \lim_{p_i \to \infty} \lambda_i = \alpha_i, \tag{3.8}$$

In addition, its supremum is given by

$$\sup \lambda_i = \frac{W(-e^{-\alpha_i \mu_i - 1}) + 1}{-\mu_i}, \tag{3.9}$$

which is attained when $p_i = 1$ and $W(\cdot)$ is the Lambert W function[130].

For better readability, we move the proofs of all lemmas, theorems and corollaries to the Appendix.

From Lemma 1, we can derive that the condition $t \geq \max_i\{\alpha_i \ell_i(t)\}$ holds, as $t = \ell_i^*(t)\lambda_i \geq \ell_i^*(t)\alpha_i$ for each work node i.

Next, we solve $\mathscr{P}_{\text{alt}}^{(2)}$. Since this problem is also NP-hard, we here provide an approximated solution. In particular, we approximate its optimal solution, denoted as t^*, with value τ^*, such that the expected amount of results accumulated at the master node by time τ^* equals to the amount of results required for recovering the final result, i.e., $\mathbb{E}[S^*(\tau^*)] = r$. To find the value of τ^*, we let

$$\mathbb{E}[S^*(t)] = r. \tag{3.10}$$

Then, using the load allocation $\ell_i^*(t)$ in Eq. (3.6), the expected amount of results received by the master node is:

$$\begin{aligned} \mathbb{E}[S^*(t)] &= \sum_{i=1}^{N} \left(\ell_i^*(t) - \frac{\ell_i^*(t)}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i(\frac{tp_i}{k\ell_i^*(t)} - \alpha_i)} \right) \\ &= \sum_{i=1}^{N} \frac{t}{\lambda_i} \left(1 - \frac{1}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i(\frac{\lambda_i p_i}{k} - \alpha_i)} \right). \end{aligned} \tag{3.11}$$

We can then find the solution to Eq. (3.10) as follows:

$$\tau^* = \frac{r}{\beta} \tag{3.12}$$

where

$$\beta = \sum_{i=1}^{N} \frac{1}{\lambda_i} \left(1 - \frac{1}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i(\frac{\lambda_i p_i}{k} - \alpha_i)} \right), \tag{3.13}$$

which is also a constant.

Combining the solutions to $\mathscr{P}_{\text{alt}}^{(1)}$ and $\mathscr{P}_{\text{alt}}^{(2)}$, we can then derive the load allocation:

$$\ell_i^*(\tau^*) = \frac{r}{\beta \lambda_i} \tag{3.14}$$

Algorithm 1: BPCC

Input: $r, N, p = \{p_1, \dots, p_N\}, \mu = \{\mu_1, \dots, \mu_N\}, \alpha = \{\alpha_1, \dots, \alpha_N\}$
Output: ℓ

1 **for** $i = 1 : N$ **do**
2   Calculate λ_i by solving Eq. (3.7)
3 Calculate β by using Eq. (3.13)
4 **for** $i = 1 : N$ **do**
5   Calculate ℓ_i^* by using Eq. (3.14)
6 **Return** $\ell = \{\lfloor \ell_1^* \rceil, \lfloor \ell_2^* \rceil, \cdots, \lfloor \ell_N^* \rceil\}$

The procedures of BPCC are summarized in Algorithm 1.

3.4.5 Optimality Analysis

In this sub-section, we conduct theoretical analysis to investigate the performance of BPCC. Specifically, we first show in Lemma 2 the optimality of the approximated solution τ^* to $\mathscr{P}_{\text{alt}}^{(2)}$. We then show in Theorem 3 that the solution provided by BPCC is asymptotically optimal. Finally, we show in Theorem 4 the accuracy of τ^* in approximating the expected execution time of BPCC.

Lemma 2. *Let t^* be the optimal solution to $\mathscr{P}_{alt}^{(2)}$, and τ^* be the approximated solution given by Eq. (3.12). If the batch processing time follows the shifted exponential distribution in Eq. (3.2) and $r = \Theta(N)$, then*

$$\tau^* - o(1) < t^* \le \tau^* + o(1). \tag{3.15}$$

Based on Lemma 2, we next show the asymptotic optimality of BPCC in Theorem 3.

Theorem 3. *Consider problem $\mathscr{P}'_{\text{main}}$ with the batch processing time following the shifted exponential distribution in Eq. (3.2) and $r = \Theta(N)$. Let $\mathbb{E}[T_{\text{BPCC}}]$ and $\mathbb{E}[T_{\text{OPT}}]$ be the expected execution time of BPCC and the optimal value of $\mathscr{P}'_{\text{main}}$, respectively. The BPCC is asymptotically optimal, i.e.,*

$$\lim_{N\to\infty} \mathbb{E}[T_{\text{BPCC}}] = \lim_{N\to\infty} \mathbb{E}[T_{\text{OPT}}] \tag{3.16}$$

Theorem 3 and Lemma 2 further lead to the following theorem.

Theorem 4. *Let τ^* be the approximated solution given by Eq. (3.12) and $\mathbb{E}[T_{\text{BPCC}}]$ be the expected execution time of BPCC. If the batch processing time follows the shifted exponential distribution in Eq. (3.2) and $r = \Theta(N)$, then*

$$\tau^* = \lim_{N \to \infty} \mathbb{E}\left[T_{\text{BPCC}}\right] \tag{3.17}$$

3.4.6 Analysis of the Impact of Parameter p

In the BPCC scheme shown in Algorithm 1, we note that p is the only parameter that can be tuned, while the other parameters, including r, N, u and α, are determined by the specific computation task and properties of the distributed computing system. In this sub-section, we analyze the impact of this important parameter p on the performance of BPCC in Theorem 5. We then show in Theorem 6 that the approximated execution time of BPCC, i.e., τ^* given by Eq. (3.12), has closed-form infimum and supremum.

Theorem 5. *Consider problem $\mathscr{P}'_{\text{main}}$ with the batch processing time following the shifted exponential distribution in Eq. (3.2) and $r = \Theta(N)$. Let τ^* be the approximated execution time of BPCC given by Eq. (3.12). Then the increase of any p_i, $i \in [N]$, will cause τ^* to decrease.*

Theorem 6. *Consider problem $\mathscr{P}'_{\text{main}}$ with the batch processing time following the shifted exponential distribution in Eq. (3.2) and $r = \Theta(N)$. Let τ^* be the approximated execution time of BPCC given by Eq. (3.12). Then*

$$\begin{aligned} \inf \tau^* = & \lim_{p_i \to \infty, \forall i \in [N]} \tau^* \\ = & \frac{r}{\sum_{i=1}^{N} \frac{1}{\alpha_i}\left(1 - e^{\mu_i \alpha_i} \int_0^1 e^{-\frac{\mu_i \alpha_i}{x}} dx\right)}, \end{aligned} \tag{3.18}$$

and

$$\sup \tau^* = \max \tau^*$$
$$= \sum_{i=1}^{N} \frac{1}{\sup \lambda_i} \left(1 - e^{-\mu_i(\sup \lambda_i - \alpha_i)}\right), \tag{3.19}$$

which is attained when $p_i = 1$, $\forall i \in [N]$. Here $\sup \lambda_i$ is given by Eq. (3.9).

From Theorem 6 and Eq. (3.14), we can derive the following corollary.

Corollary 6.1. *Consider problem $\mathscr{P}'_{\text{main}}$ with the batch processing time following the shifted exponential distribution in Eq. (3.2) and $r = \Theta(N)$. Let ℓ_i^* be the solution of BPCC given by Eq. (3.14). Then when the approximated execution time τ^* of BPCC given by Eq. (3.12) converges to its infimum, ℓ_i^* converges to $\hat{\ell}_i$, where*

$$\hat{\ell}_i = \frac{r}{\alpha_i \sum_{j=1}^{N} \frac{1}{\alpha_j}(1 - e^{\mu_j \alpha_j} \int_0^1 e^{-\frac{\mu_j \alpha_j}{x}} dx)}. \tag{3.20}$$

3.4.7 Comparison with HCMM

In this sub-section, we compare the performance of BPCC with HCMM [25], a state-of-the-art CDC scheme for heterogeneous worker nodes, and show that BPCC outperforms HCMM in computational efficiency.

HCMM can be considered as a special case of BPCC with $p_i = 1$, $\forall i \in [N]$. It assigns each worker node i with load $\ell_{H,i} = \frac{r}{\beta_H \lambda_{H,i}}$, where $\lambda_{H,i}$ is the positive solution to $e^{\mu_i \lambda_{H,i}} = e^{\alpha_i \mu_i}(\mu_i \lambda_{H,i} + 1)$ and $\beta_H = \sum_{i=1}^{N} \frac{\mu_i}{1+\mu_i \lambda_{H,i}}$. Theorem 7 shows that BPCC is more efficient than HCMM.

Theorem 7. *Consider problem $\mathscr{P}'_{\text{main}}$, with the batch processing time following a shifted exponential distribution in Eq. (3.2) and $r = \Theta(N)$. Let T_{BPCC} and T_{HCMM} be the execution times*

of BPCC and HCMM, respectively. Then,

$$\lim_{N\to\infty} \mathbb{E}[T_{\text{BPCC}}] \leq \lim_{N\to\infty} \mathbb{E}[T_{\text{HCMM}}]$$

3.5 Simulation Studies

In this section, we conduct simulation studies to evaluate the performance of the proposed BPCC scheme. Specifically, we first explain the simulation settings, including the distributed computing schemes and scenarios. We then elaborate on the impact of important parameters, including p_i, N, μ_i and α_i, on the performance of the BPCC scheme. Finally, we compare the proposed BPCC scheme with benchmark schemes, including the state-of-the-art HCMM scheme [25].

3.5.1 Simulation Settings

Distributed Computing Schemes

In this study, we consider four distributed computing schemes:

- **Uniform Uncoded:** This method divides the computation loads equally, i.e., $\ell_i = \frac{r}{N}$, $\forall i \in [N]$.
- **Load-Balanced Uncoded [25]:** This method divides the computation loads according to the computing capabilities of the worker nodes. In particular, the computation load assigned to each worker node i is inversely proportional to the expected time for this node to compute an inner product, i.e., $\ell_i \propto (\frac{\mu_i}{\mu_i\alpha_i+1})$ and $\sum_{i=1}^{N} \ell_i = r$.
- **HCMM [25]:** In this method, the load assignment method in [25] is used. Note that this is a special case of Algorithm 1, in which $p_i = 1, \forall i \in [N]$. The HCMM and BPCC have the exactly same load allocation for each worker.
- **BPCC:** In this scheme, Algorithm 1 is used, where p are the parameters to configure.

Computation Scenarios

To evaluate the performance of different distributed computing schemes, we consider the following four computation scenarios:

- **Scenario 1:** $r = 1 \times 10^4$ and $N = 10$.
- **Scenario 2:** $r = 2 \times 10^4$ and $N = 10$.
- **Scenario 3:** $r = 1 \times 10^4$ and $N = 20$.
- **Scenario 4:** $r = 2 \times 10^4$ and $N = 20$.

Simulation Method

In our simulation, we implement all the aforementioned distributed computing schemes in MATLAB. We assume that the processing time of each node follows the shifted exponential distribution in Eq. (3.2). Specifically, for each experiment of a scenario, we choose the straggling parameters $\mu_i, \forall i \in [N]$ randomly in $[1, 50]$, and calculate each shift parameter $\alpha_i = \frac{1}{\mu_i}$. In each experiment, we simulate every distributed computing scheme for 100 times, in each of which the computing time of a node is simulated by using its straggling and shift parameters.

3.5.2 Parameter Impact Analysis

In this sub-section, we investigate the impacts of parameters in BPCC, including number of batches, number of worker nodes, and the straggling and shift parameters in the computing model.

Number of batches

The number of batches p_i is an important parameter to configure. In Section 3.4.6, we have theoretically analyzed its impact on the performance of BPCC. Here we conduct simulation studies to demonstrate its impact described in Theorem 5, Theorem 6 and Corollary 6.1. In particular, two experiments are designed.

In the first experiment, we show that the approximated execution time τ^* of BPCC given by Eq. (3.12) decreases with the increase of any p_i, $i \in [N]$, as presented in Theorem 5. In particular, we vary the number of batches for one of the worker nodes and fix the number of batches for the others. Specially, we vary p_1 and let $p_j = 1, \forall j \in [N] \setminus \{1\}$. As shown in Fig. 3.2(a), τ^* indeed decreases as p_1 increases.

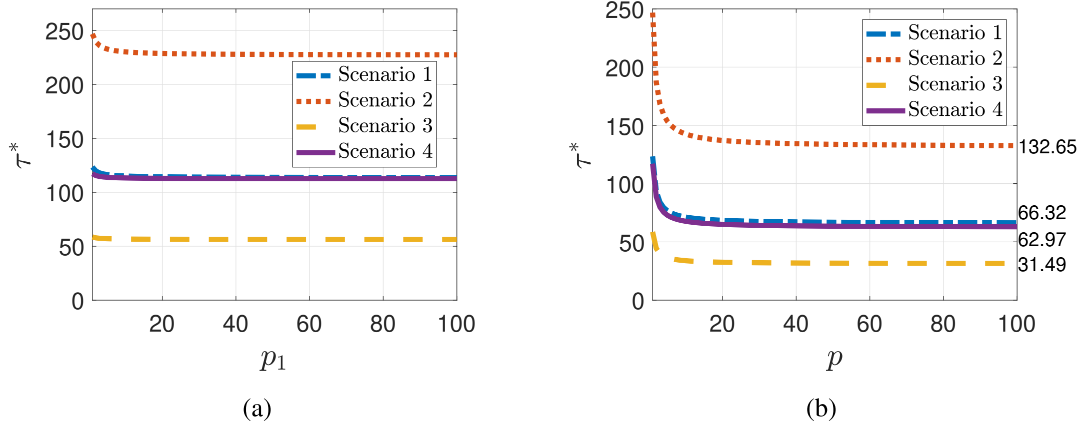


Figure 3.2. The approximated execution time τ^* of BPCC at different values of a) p_1, when $p_j = 1, \forall j \in [N] \setminus \{1\}$, and b) p, when $p_i = p, \forall i \in [N]$, in different scenarios.

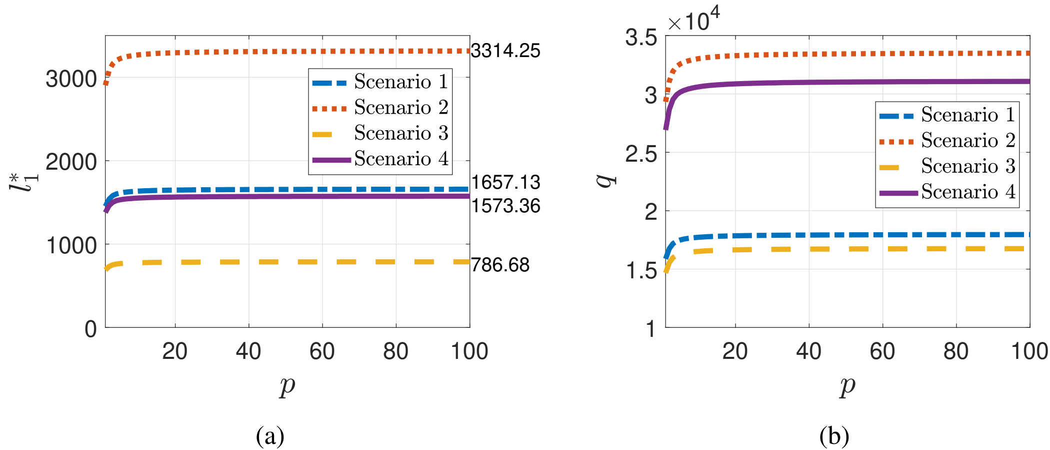


Figure 3.3. The value of a) load ℓ_1^* and b) total load $q = \sum_{i=1}^{N} \ell_i^*$ at different values of p, when $p_i = p, \forall i \in [N]$, in different scenarios.

In the second experiment, we show that the approximated execution time τ^* and the load ℓ_i^* tend to converge as p_i increases for all $i \in [N]$, as presented in Theorem 6 and Corollary 6.1.

In this experiment, we vary p_i simultaneously for all $i \in [N]$. In other words, we let $p_i = p \in \mathbb{Z}^+$, $\forall i \in [N]$ and vary the value of p. As shown in Fig. 3.2(b), τ^* decreases with the increase of p and finally converges. Note that when $p = 100$, τ^* equals to 66.32, 132.65, 31.49, 62.97 for the four scenarios, respectively, which are already very close to its theoretical infimum 65.77, 131.54, 31.22, 62.45, computed by Eq. (3.18). Fig. 3.3(a) shows the trajectory of the load allocated to one of the worker nodes, i.e., ℓ_1^*, which decreases and finally converges as p increases. Note that when $p = 100$, ℓ_1^* equals to 1657.13, 3314.25, 786.68, 1573.36 for the four scenarios, respectively, which are very close to the values of $\hat{\ell}_1$ given by Eq. (3.20), i.e., 1659.79, 3319.59, 787.95, 1575.90. Of interest, if we set $p_i = \lfloor \hat{\ell}_i \rfloor$ given by Eq. (3.20), $\forall i \in [N]$, τ^* equals to 65.81, 131.58, 31.26, 62.47 and ℓ_1^* equals to 1659.62, 3319.42, 787.73, 1575.68 for the four scenarios, respectively, which are almost the same as the associated $\inf \tau^*$ and $\hat{\ell}_1$, respectively.

Fig. 3.3(a) shows the impact of parameter p_i on the load ℓ_i^* for one of the work nodes. In Fig. 3.3(b), we also show its impact on the total load $q = \sum_{i=1}^{N} \ell_i^*$. As we can see, the total load q also increases with the increase of p, where $p_i = p$, $\forall i \in [N]$. This indicates that a larger p_i will require more storage space at the worker nodes. Note that the worker nodes will stop execution once the master node receives sufficient amount of results for recovering the final result. Therefore, a larger total load q does not increase the computation load for the worker nodes. This study tells us that the configuration of parameter p_i should trade off between computational efficiency and storage consumption.

As τ^* is an approximation of BPCC's execution time, we also show in Fig. 3.4 the impact of p_i on the expected execution time $\mathbb{E}[T_{\text{BPCC}}]$ of BPCC, which is estimated using the Monte Carlo simulation method, specifically, by repeating each experiment for 100 times and averaging the times to execute the BPCC scheme. Comparing Fig. 3.2 and Fig. 3.4, we can see that τ^* approximates $\mathbb{E}[T_{\text{BPCC}}]$ generally well. The fluctuations are caused by the uncertainty of the computation times and the relatively weak estimation capability of the Monte Carlo method, which requires large number of simulations to obtain an accurate mean estimate. As we will show in the next study, the approximation accuracy of τ^* is impacted by the number of worker

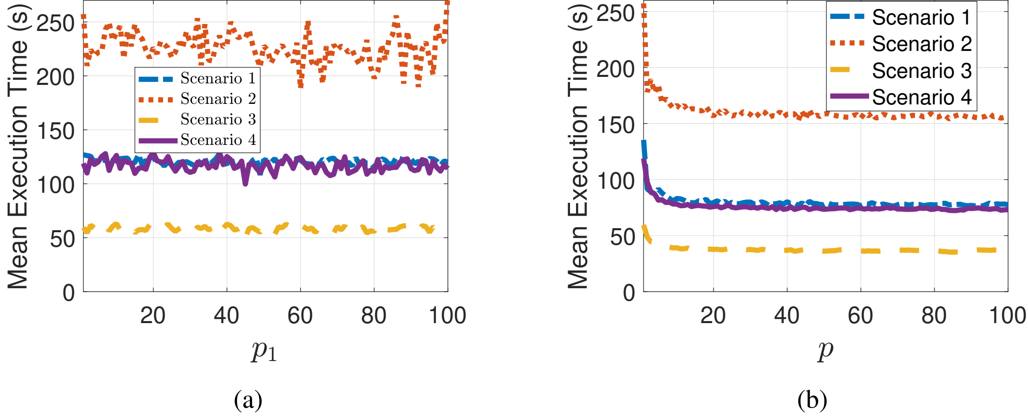


Figure 3.4. The mean execution time of BPCC at different values of a) p_1, when $p_j = 1$, $\forall j \in [N] \setminus 1$, and b) p, when $p_i = p, \forall i \in [N]$, in different scenarios.

nodes N.

Number of worker nodes

As we have theoretically proved in Theorem 4, the approximated execution time τ^* converges to the true expected execution time $\mathbb{E}[T_{\text{BPCC}}]$ of BPCC, when the number of worker nodes N approaches infinity. To demonstrate this theorem, we vary N and set $r = 100N + 10000$, and record the approximation error of τ^*, given by $|\tau^* - \mathbb{E}[T_{\text{BPCC}}]|$, for each value of N. The results are shown in Fig. 3.5. Note that, instead of the four scenarios described in Section 3.5.1, we design four new scenarios for this study, where the configuration of each scenario is specified in the figure. As we can see, the approximation error of τ^* decreases with the increase of the number of worker nodes, and finally converges to zero.

From the above studies, we can see that, as the number of batches p_i for any worker node $i \in [N]$ increases, the efficiency of BPCC improves, but the demand for storage also increases. Because storage consumption is not our main concern in this study, in the following experiments, we set p_i to its maximum value possible, i.e., $p_i = \lfloor \hat{\ell}_i \rfloor$, $\forall i \in [N]$, considering that a valid p_i should be a positive integer smaller than or equal to ℓ_i^* and ℓ_i^* converges to $\hat{\ell}_i$ as p_i, $\forall i \in [N]$, increases.

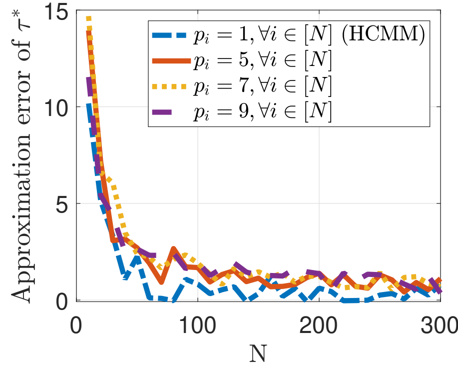


Figure 3.5. The approximation error of τ^* at different values of N with $r = 100N + 10000$.

Straggling and shift parameters

In BPCC, to determine the load numbers ℓ_i, we need to know the values of the straggling and shift parameters, μ_i and α_i, which are estimated by measuring the actual execution behaviors in real experiments. To understand the impact of parameter estimation errors to the performance of BPCC, we conduct a sensitivity study. In particular, to study how sensitive BPCC is to the estimation errors associated with the straggling parameters μ_i, we fix the shift parameters α_i and deviate each μ_i from its true value by randomly picking a value from the interval $(\mu_i^{min}, \mu_i^{max})$, where $\mu_i^{min} = \mu_i^*(1-\Delta)$, $\mu_i^{max} = \mu_i^*(1+\Delta)$, μ_i^* is the true value and $\Delta > 0$ represents the degree of deviation. As μ_i should be positive, we let $\mu_i^{min} = 0$, if $\Delta > 1$. Fig. 3.6(a) shows the relative change of the mean execution time, measured by $\frac{\hat{\mathbb{E}}'[T]-\hat{\mathbb{E}}[T]}{\hat{\mathbb{E}}[T]}$, at different values of Δ in different scenarios, where $\hat{\mathbb{E}}[T]$ and $\hat{\mathbb{E}}'[T]$ are the mean execution time obtained by using the true and erroneous parameter values, respectively. Similarly, we plot in Fig. 3.6(b) the relative change of the mean execution time when the shift parameters α_i suffer from estimation errors. As we can see, the deviation of straggling parameters μ_i has less impact on the performance of BPCC than that of shift parameters α_i, and BPCC is robust to small errors in general.

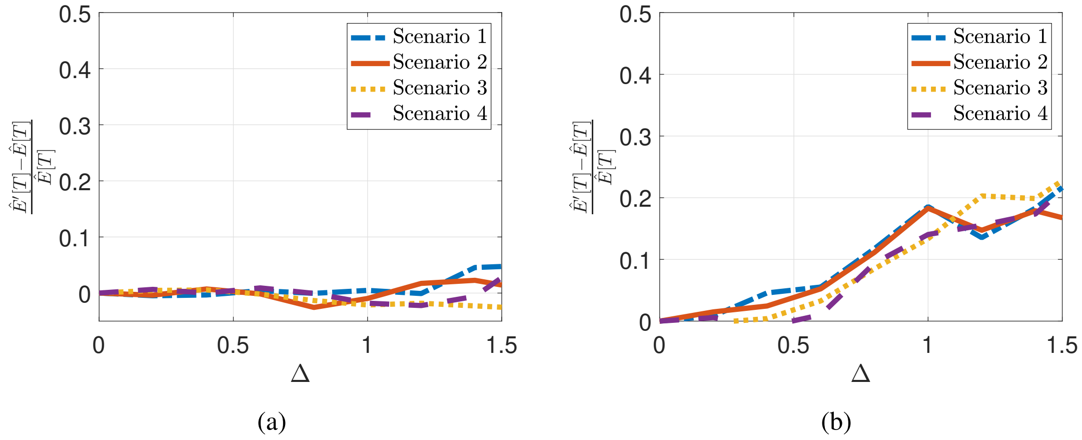


Figure 3.6. Relative change of the mean execution time when a) the straggling parameters μ_i and b) the shift parameters α_i suffer from different degrees of deviation from their true values in different scenarios.

3.5.3 Comparative Performance Studies

In this sub-section, we compare the performance of the proposed BPCC scheme with three benchmark schemes, including Uniform Uncoded, Load-Balanced Uncoded and HCMM. The parameter p_i in BPCC is set to $p_i = \lfloor \hat{\ell}_i \rfloor, \forall i \in [N]$.

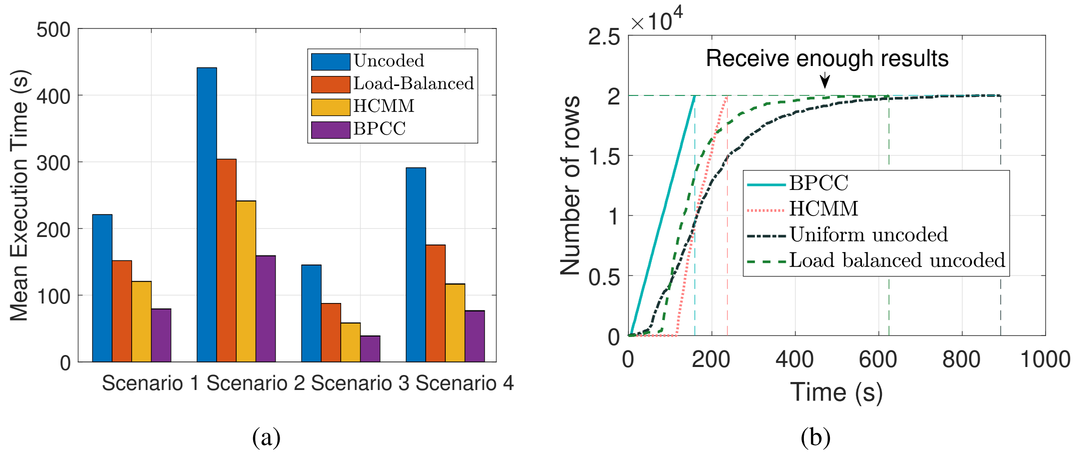


Figure 3.7. a) Comparison of the mean execution time of different schemes in different scenarios. b) The average total number of rows of inner product results received by the master node over time for different schemes in Scenario 2.

Fig. 3.7(a) shows the mean execution times for all schemes, grouped by the computation scenario. We can clearly observe that the proposed BPCC scheme outperforms other benchmark schemes in all scenarios. For instance, BPCC achieves performance improvement of up to 73% over the Uniform Uncoded scheme, up to 56% over Load-Balanced Uncoded scheme, and up to 34% over HCMM. Note that the execution times are directly derived by using the computing model in Eq. (3.2) and the decoding times are not considered here.

In Fig. 3.7(a), the performance is expressed in terms of the mean execution time, which corresponds to $\mathbb{E}[T_{\text{BPCC}}]$ for different schemes. In Fig. 3.7(b), we show the average amount of received results over time for Scenario 2, which corresponds to $\mathbb{E}[S(t)]$ in the theoretical analysis. Remarkably, we can observe from the figure that the master node can quickly receive results from the worker nodes from the very beginning. On the other hand, under the three benchmark schemes, there is a certain duration at the beginning when the master node does not receive any result. This phenomenon occurs because our BPCC scheme allows partial results to be returned, which is very useful for certain applications that can utilize partial results. In Fig. 3.7(b), we also indicate the time when the master node receives the required amount of results, i.e., r. Such a time corresponds to τ^* (i.e., $\mathbb{E}[S(\tau^*)] = r$).

3.6 Experiments on the Amazon EC2 Computing Cluster

In this section, we evaluate the performance of the proposed BPCC scheme in the real distributed computing system. Specifically, we implement three benchmark schemes and the proposed BPCC scheme in the Amazon EC2 computing platform [131], which is a classical cloud computing system.

3.6.1 Experiment Settings

To implement the proposed BPCC and the three benchmark schemes over Amazon EC2 clusters [1], we apply a standard distributed computing interface, *Message Passing Interface* (MPI)

[1]The source code can be found at https://github.com/BaoqianWang/Batch-Processing-Coded-Computation.

[132], by using an open-source package: `mpi4py` [133], which provides interfaces in Python. Moreover, to encode and decode matrices in BPCC, we use the Luby Transform (LT) codes with peeling decoder [129] that are adopted by HCMM [25]. The utilization of LT code relaxes the constraint of recovering the final computation result from any r rows to any $r(1+\varepsilon)$ rows, where $\varepsilon > 0$ is desired to be as small as possible. In this study, we adopt the configuration in [25] and set $\varepsilon = 0.13$. The parameter p_i in BPCC is set to $p_i = \lfloor \hat{\ell}_i \rfloor$, $\forall i \in [N]$.

To evaluate the performance of the four computation schemes, we consider the following four scenarios:

- **Scenario 1:** $r = 0.5 \times 10^4$ and $N = 5$, where one *r4.2xlarge* instance, two *r4.xlarge* instances and two *t2.large* instances are used as the worker nodes.
- **Scenario 2:** $r = 1 \times 10^4$ and $N = 10$, where two *r4.2xlarge instances* instances, four *r4.xlarge* instances, and four *t2.large* instances are used as the worker nodes.
- **Scenario 3:** $r = 1.5 \times 10^4$ and $N = 10$, where four *r4.2xlarge* instance and six *r4.xlarge* instances are used as the worker nodes.
- **Scenario 4:** $r = 2 \times 10^4$ and $N = 15$, where seven *r4.2xlarge* instance and eight *r4.xlarge* instances are used as the worker nodes.

In all above scenarios, the master node runs in a *m4.xlarge* instance, and the size of the input vector $x \in \mathbb{R}^m$ is set to $m = 5 \times 10^5$.

3.6.2 Parameter Estimation

In our previous design and analysis, we have assumed that the task completion time T on each node follows a shifted exponential distribution in a general form:

$$\Pr[T \leq t] = 1 - e^{-\mu(\frac{t}{r} - \alpha)} = 1 - e^{-\frac{\mu}{r}(t - \alpha r)}, \tag{3.21}$$

when $t \geq \alpha r$. Therefore, $\mathbb{E}[T] = \frac{r}{\mu} + \alpha r$.

Based on the assumption above, we conduct extensive experiments and measure the actual execution behaviors to estimate the values of the straggling and shift parameters, μ and α, for different types of instances. Particularly, let $t_c(r) = \frac{r}{\mu}$ and $t_0(r) = \alpha r$, we run tasks of different sizes. For each task size r, we execute the task repeatedly for $M = 1000$ times and obtain the execution times $\{T_1, T_2, \ldots, T_M\}$. The maximum likelihood estimates of $t_0(r)$ and $t_c(r)$ are then given by $\hat{t}_0(r) = \min_{l \in [M]} T_l$ and $\hat{t}_c(r) = \frac{1}{M} \sum_{l=1}^{M} T_l - \hat{t}_0(r)$, respectively [134, 135]. With $\hat{t}_0(r)$ and $\hat{t}_c(r)$ for different task sizes r, we can then estimate the values of μ and α by using the least squares estimation. Fig. 3.8 shows the estimated *cumulative distribution function* (CDF) of the processing time of a t2.xlarge instance when task size $r = 500$. The estimated α and μ for different types of Amazon EC2 instances are summarized in Table 3.2. These estimated parameters will be used to allocate computation loads for all computing schemes, except the uniform uncoded scheme.

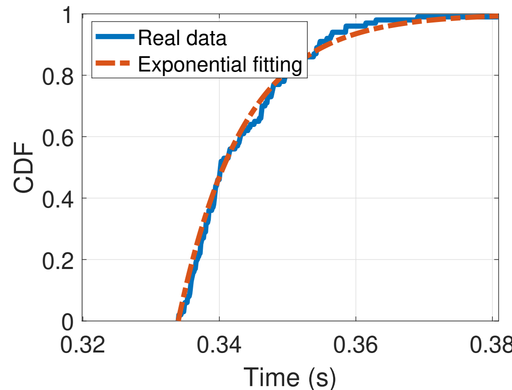


Figure 3.8. The CDF of the processing time of an Amazon EC2 t2.xlarge instance for computing a task with $r = 500$.

3.6.3 Experimental Results

To evaluate the performance of the proposed BPCC scheme running on the heterogeneous Amazon clusters, we design three experiments.

Table 3.2. Estimated computing parameters of different types of Amazon EC2 instances

Instances	μ	α
r4.xlarge	9.42×10^4	1.75×10^{-4}
r4.2xlarge	9.25×10^4	1.60×10^{-4}
t2.medium	2.15×10^4	5.18×10^{-4}
t2.large	3.90×10^4	2.25×10^{-4}

Experiment 1

In this experiment, we compare the performance of BPCC with the three benchmark schemes in different scenarios. For each scenario, we run each scheme 100 times and record the mean execution time ($\mathbb{E}[T]$). To evaluate the robustness of these schemes to uncertain disturbances, we introduce unexpected stragglers that are randomly chosen in each run. In particular, we randomly select 20% of the worker nodes to be stragglers in each run. As stragglers can be slow in computing or returning results (e.g, when communication congestion happens), such stragglers are emulated by delaying the return of computing results such that the computing time observed by the master node is three times of the actual computing time. Fig. 3.9(a) illustrates the mean execution time of different distributed computing schemes in different scenarios, which also highlights the decoding time required by the coded schemes including BPCC and HCMM. We can see from the figure that the proposed BPCC scheme outperforms all benchmark schemes in all scenarios. Specifically, the performance improves up to 79% compared with Uncoded scheme, up to 78% compared with Load-Balanced scheme, and up to 62% compared with HCMM.

As stragglers can also fail in returning any results (e.g., when nodes/links fail), we also consider such stragglers and emulate them by setting the delay time to infinity. Since no results will be returned by such stragglers, the computation task can fail. Fig. 3.9(b) shows the success rate (measured by the ratio of successful runs) of each scheme in different scenarios. The mean execution time of successful runs is shown in Fig. 3.9(c). As we can see, the Uncoded and Load-Balanced schemes fail to complete the task in all runs, as no redundancy is introduced in

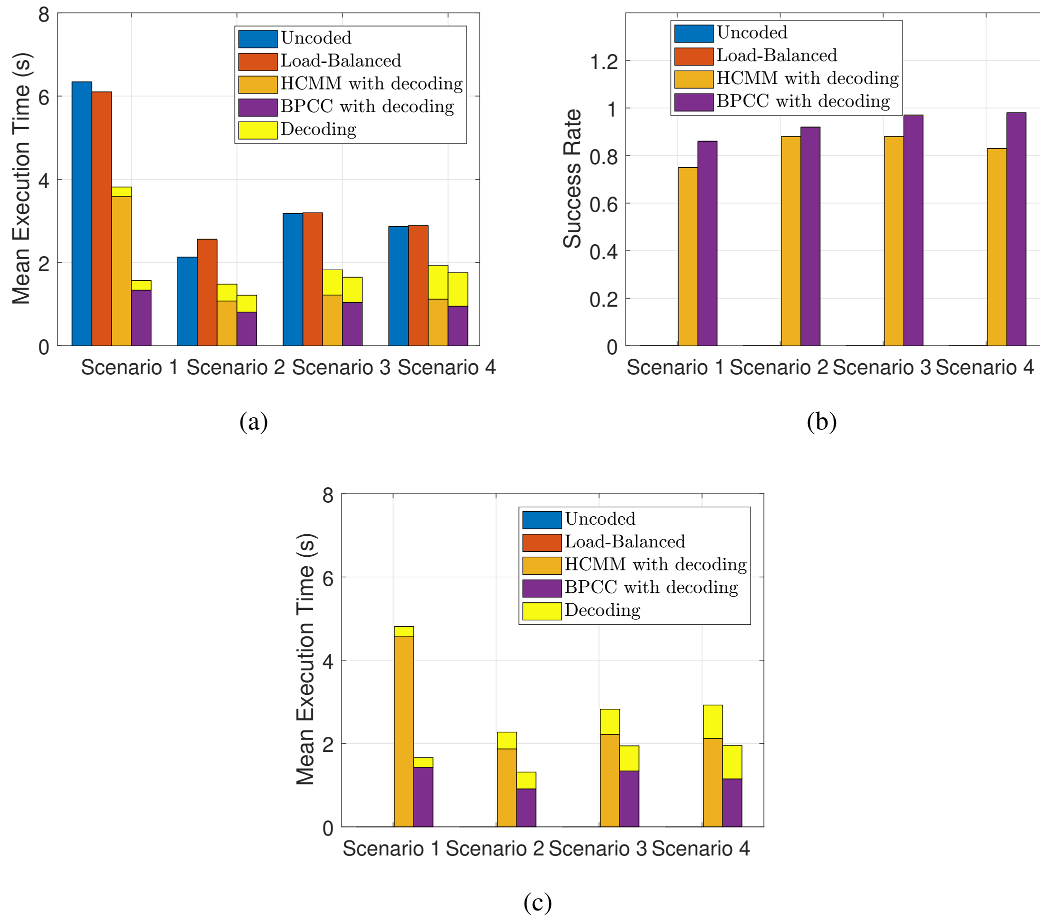


Figure 3.9. a) The mean execution time of different schemes in different scenarios at the presence of unexpected stragglers with finite delay. The b) success rate and c) mean execution time of different schemes in different scenarios at the presence of unexpected stragglers with infinite delay.

these schemes. Both HCMM and BPCC can successfully complete the task in most runs, but HCMM has a lower success rate and is less efficient than BPCC.

In Fig. 3.10, we selectively show the average amount of received results over time ($\mathbb{E}[S(t)]$) in Scenario 4. As expected, in our BPCC scheme, the master node continuously receives results from the very beginning. However, in other schemes, the master node needs to wait a long time before receiving any result.

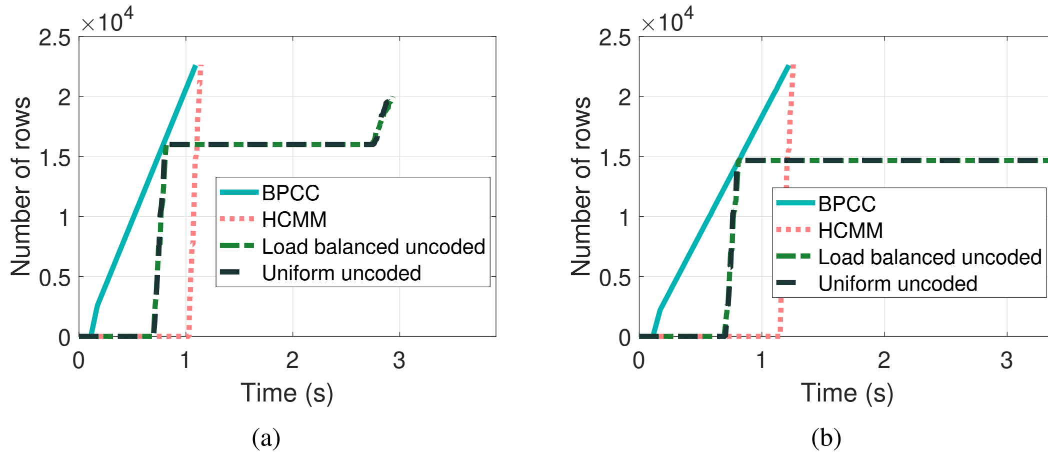


Figure 3.10. The average total number of rows of inner product results received by the master node over time for different schemes in Scenario 4 at the presence of unexpected stragglers with a) finite and b) infinite delay.

Experiment 2

In the second experiment, we study the impact of the number of unexpected stragglers on the performance of the four computation schemes, by varying the percentage of stragglers from 0% to 60%. Similar as Experiment 1, stragglers can delay in returning results for finite or infinite amount of time.The mean execution time of each scheme at the presence of different numbers of stragglers with finite delay for Scenario 4 is shown in Fig. 3.11(a). As we can see, when there is no straggler, the Uniform Uncoded scheme and the Load-Balanced Uncoded scheme achieve the best performance, as they do not involve any computation redundancy, compared with the coded schemes. However, when stragglers exist, our BPCC scheme achieves the best performance, indicating its high robustness to uncertain stragglers. We can also observe from Fig. 3.11(a) that the performances of all schemes degrade with the increase of the number of stragglers. Of interest, the performance degradation of the three benchmark schemes slows down when the number of stragglers reaches to a certain value. This is because worker nodes in these schemes won't return any result to the master node until the whole assigned task is completed and all stragglers would delay returning the result for a period that is three times of the task computation time. We also note that the performance of HCMM is even worse than the two uncoded schemes

when the percentage of stragglers exceeds 20%. This is because each worker node in HCMM is assigned with more computation load, compared with the uncoded schemes, which causes the stragglers in HCMM to wait for a longer time before returning any result.

In case when stragglers delay in returning results for infinite amount of time, the success rate and the mean execution time of each scheme are shown in Fig. 3.11(b) and Fig. 3.12(a), respectively. As expected, both the Uncoded and Load-Balanced schemes fail to complete the task. Additionally, the performances of BPCC and HCMM degrade with the increase of the number of stragglers, and BPCC outperforms HCMM.

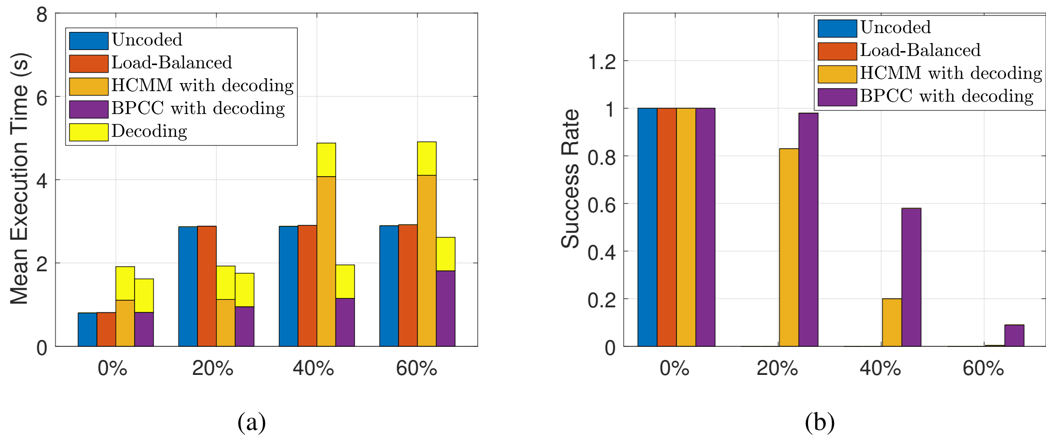


Figure 3.11. a) The mean execution time of different schemes in Scenario 4 when different percentages of unexpected stragglers with finite delay are present. b) The success rate of different schemes in Scenario 4 when different percentages of unexpected stragglers with infinite delay are present.

Experiment 3

In the third experiment, we evaluate the impact of the number of batches p_i on the performance of BPCC running on the Amazon clusters. Similar as the simulation study, we let $p_i = p$, $\forall i \in [N]$, and vary the value of p from 5 to 100. Fig. 3.12(b) shows the mean execution time of BPCC at different values of p under the settings described in Scenario 4 and Experiment 1 when unexpected stragglers with finite delay are present. As expected, the efficiency of BPCC

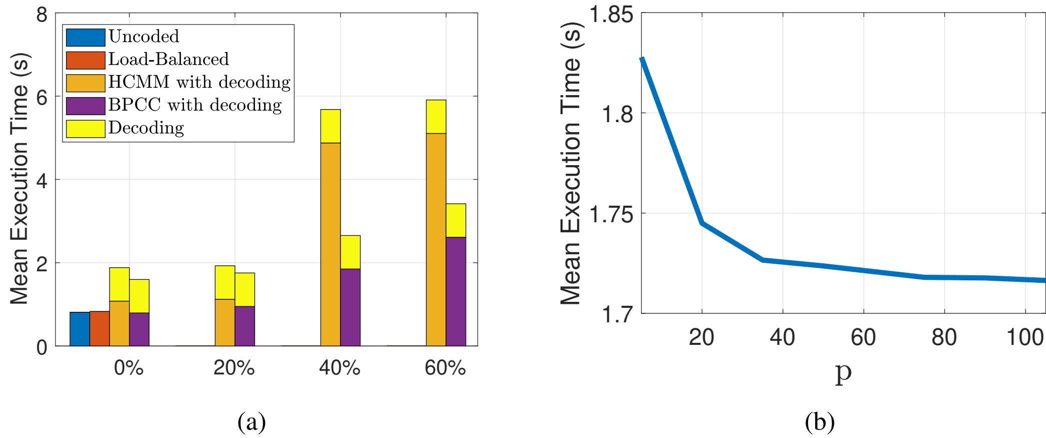


Figure 3.12. The a) mean execution time of different schemes in Scenario 4 when different percentages of unexpected stragglers with infinite delay are present. b) The mean execution time of BPCC at different values of p in Scenario 4.

improves with the increase of p.

3.7 Conclusion

In this chapter, we systematically investigated the design and evaluation of a novel *coded distributed computing* (CDC) framework, namely, *batch-processing based coded computing* (BPCC), for heterogeneous computing systems. The key idea of BPCC is to optimally exploit partial coded results calculated by all distributed computing nodes. Under this BPCC framework, we then investigated a classical CDC problem, matrix-vector multiplication, and formulated an optimization problem for BPCC to minimize the expected task completion time, by configuring the computation load. The BPCC was proved to provide an asymptotically optimal solution and outperform a state-of-the-art CDC scheme for heterogeneous clusters, namely, *heterogeneous coded matrix multiplication* (HCMM). Theoretical analysis reveals the impact of BPCC's key parameter, i.e., number of batches, on its performance, the results of which infer the worst and best performance that BPCC can achieve. To evaluate the performance of the proposed BPCC scheme and better understand the impacts of its parameters, we conducted extensive

simulation studies and real experiments on the Amazon EC2 computing clusters. The simulation and experimental results verify theoretical results and also demonstrate that the proposed BPCC scheme outperforms all benchmark schemes in computing systems with uncertain stragglers, in terms of the task completion time and robustness to stragglers. In the future, we will further enhance BPCC by jointly optimizing load allocation and the number of batches to achieve a tradeoff between computational efficiency and storage consumption, and explore its other properties, such as the convergence rate.

3.8 Acknowledgement

This Chapter is a reprint of the published journal: B. Wang, J. Xie, K. Lu, Y. Wan, S. Fu, "On Batch-Processing Based Coded Computing for Heterogeneous Distributed Computing Systems", *IEEE Transactions on Network Science and Engineering*, Vol.8, pp:2438-2454, 2021.

Chapter 4

Learning and Batch-Processing Based Coded Computation for Mobile Networked Airborne Computing Systems

4.1 Introduction

In the previous chapter, we developed a Batch-Processing Based Coded Computation distributed computing framework for static and heterogeneous distributed computing systems, which is applicable to NAC formed by hovering. This chapter extends the work of Chapter 2 to consider the movements of UAVs.

When UAVs operate in complex and uncertain airspace with high mobility, the fast node movement, line-of-sight effect, and node leaving and joining can cause frequent topology changes, link failures, data losses, and task interruptions. Moreover, the various uncertainties (e.g., winds and other vehicles) present in the airspace can disturb the communication among the UAVs, bringing additional challenges for robust computing. This chapter aims to tackle these key technical challenges of NAC to achieve robust cooperative airborne computing onboard of multiple networked UAVs. In our study, we consider that a NAC system can be formed in different ways, and there are two realistic formation scenarios. First, the NAC system is formed by UAVs operated by different owners in an opportunistic manner, e.g., when cargo drones owned by different companies are serving the same area. Here, the mobility of the UAVs is

uncontrollable, unknown, and can be considered random. Second, the NAC system is formed by UAVs operated by the same owner, e.g., in multi-UAV applications like multi-UAV surveillance, search and rescue. In this scenario, the mobility of the UAVs can be controlled and proactively planned by the owner to facilitate computing. In this study, we consider both formation scenarios and develop innovative computation schemes for the two scenarios to enable efficient, robust, and adaptable cooperative airborne computing in an uncertain, heterogeneous, and dynamic airspace. The main contributions are summarized as follows:

1. *Dynamic batch-processing based coded computation (D-BPCC) framework:* This framework features a dynamic batch-processing based procedure and applies the coding theory to address the uncertainties phenomenal in a dynamic NAC system and to improve the efficiency, robustness, and adaptability of the computing system.

2. *Deep reinforcement learning (DRL) based optimization and control:* DRL-based online decision-making strategies are designed to optimize the system performance and control UAV mobility. Compared to the conventional numerical optimization methods [11, 12, 13, 14, 15, 16, 17, 18], DRL-based strategies do not require any knowledge of the communication, computation, or UAV mobility models, can be quickly deployed in any NAC systems, and generate solutions in real time.

3. *Two typical NAC formation scenarios:* We address two typical NAC formation scenarios by applying the DRL-based and D-BPCC-based schemes. To the best of our knowledge, these two scenarios have not been investigated in the literature.

4. *Comprehensive simulation studies:* We conduct comprehensive simulation studies to evaluate the performance of the proposed methods for the two NAC formation scenarios from several aspects. We also implement four state-of-the-art distributed computing schemes as the benchmarks for comparison studies.

4.2 Related Work

In this section, we review related work in four areas: networked airborne computing (NAC), UAV-assisted mobile edge computing (MEC), DRL-based UAV-assisted networks, and coded distributed computing.

4.2.1 Networked Airborne Computing

Most existing works on UAV-assisted computing focus on MEC [11, 12, 13, 14, 15, 16, 17, 18], where UAVs function as servers to provide computing services to ground users. Studies that explore resource sharing among UAVs in uncertain airspace via direct flight-to-flight links are very limited. In [9], the concept of NAC, its advantages and design guidelines were introduced. In [78, 136], we investigated the hardware and software design for a prototype of the NAC platform. In [137], we considered a NAC system formed by static UAVs hovering in the air and developed a coded distributed computing scheme to achieve robust computation of matrix multiplication tasks.

A more general concept, called mobile ad hoc computing or mobile ad hoc cloud [138, 139, 140], was coined recently in [138], which refers to any computing systems formed by mobile devices with resources shared among each other. Existing studies have mostly considered smartphones [141, 142, 143] or ground vehicles [144] as the main computing resource providers. Nevertheless, substantial differences exist between NAC and ground-based mobile ad hoc computing due to the unique features of UAVs such as high 3-D mobility, highly uncertain operating environment with significant impacts on aerial dynamics, stringent safety requirements, mechanical and aerospace constraints. Existing solutions cannot address these new technical challenges of NAC.

4.2.2 UAV-assisted Mobile Edge Computing

MEC can address the high transmission latency of remote cloud-based computing paradigms by deploying cloud resources at the network edge close to users [10]. In UAV-assisted MEC [145], UAV with computing power functions as an edge server to provide computing services for ground users. To improve the quality of service (QoS), joint computation offloading and UAV trajectory designs were investigated in [11, 12, 13, 14, 15, 16, 17, 18], where the UAV follows an optimized trajectory for serving multiple static ground users. Recently, a few studies extended the problem to multiple UAVs [146, 147, 148]. In particular, [146, 147] made UAVs hover statically over ground users and investigated the optimal placement of UAVs and task assignment. In [148], the trajectories of UAVs were optimally designed, while jointly considering the optimal bit allocation and task assignment. In [19], the authors considered an MEC system formed by stationary and UAV-based quasi-stationary MEC servers, and investigated how to form coalitions with shared resources for MEC servers to better serve their users.

To solve the aforementioned optimization problems, most studies [11, 12, 13, 14, 15, 16, 17, 18, 146, 147, 148] assumed known and time-invariant communication and computation models and attempted to derive exact solutions. However, this assumption often does not hold in reality.

4.2.3 DRL-based UAV-Assisted Networks

To manage unknown and complex UAV-assisted networks, DRL has been explored in many recent studies [149, 23, 150, 151, 152, 153, 154]. For example, multi-agent reinforcement learning (MARL) was used in [154] to optimize the wireless energy transfer between UAVs and flying energy resources. Deep Q Network (DQN) was employed in [153] to achieve efficient dispatch of UAVs as relays in vehicular networks. In [149], DQN was used for UAV path planning in UAV-assisted MEC to minimize energy consumption and maximize task completion efficiency. In [23], MARL was applied to plan UAV trajectories, based on which an optimization

approach was developed for computation offloading. DRL has also been explored to generate offloading decisions in UAV-assisted MEC [151, 152], but it hasn't been applied to NAC.

4.2.4 Coded Distributed Computing

The resilience of distributed computing systems to uncertain system disturbances can be enhanced using the coded computation techniques. The key idea is to apply the coding theory to generate redundant computations for reducing the impact of disturbances. This idea has been explored for different computation problems, such as matrix multiplications [114, 115, 126, 155, 156, 157], linear inverse problems [158], convolution [159], deep neural networks [160], map-reduce [161], and MARL [162].

Nevertheless, most of these approaches were developed based on multiple assumptions (e.g., homogeneous and static computing nodes, known and time-invariant data transfer behavior and computing power) and thus, cannot be directly applied for NAC.

Recently, coded distributed computing (CDC) has also been explored to facilitate MEC. For example, an error-correcting-code-inspired strategy was proposed in [103] to execute computation tasks in edge servers. Paper [163] introduced a coding scheme that combines a rateless code for improving system resiliency and an irregular-repetition code for reducing the communication latency. However, both approaches assume homogeneous and static computing nodes. In [164], a Lagrange coded computing-based framework was developed to enable fast and secure computation in MEC with heterogeneous but static edge servers. Papers [126, 155] are closely related to this work, which consider a MEC system with heterogeneous and mobile computing nodes. To reduce task completion delay, a coded computation framework called the coded cooperative computation protocol (C3P) was developed. Although this framework can address the first NAC formation scenario with uncontrollable UAVs, it cannot address the second scenario that requires UAV mobility control. Moreover, as we will show in the Simulation Studies section (Sec. 6.5), C3P is vulnerable to frequent network changes caused by node movement.

4.3 NAC System

In this chapter, we consider a NAC system that consists of multiple UAVs flying at the same altitude. The system can be either formed by UAVs with random mobility, i.e., uncontrollable and unpredictable, or UAVs that can be proactively maneuvered. The onboard computation tasks to be executed by the UAVs cooperatively are assumed to be matrix-vector multiplications $\mathbf{Ax}$, where $\mathbf{A} \in \mathbb{R}^{p\times m}$ is a pre-stored matrix. Matrix-vector multiplication is considered here as it is the building block for many computation tasks, especially machine learning based applications such as collaborative filtering recommender systems [165] and object detection [166]. The proposed computation framework can be easily extended for other problems, such as convolution [167] and distributed path planning [168].

The UAV that receives a sequence of input vectors, $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\}$, to process is referred to as the *master node*, where $\mathbf{x}_j \in \mathbb{R}^{m\times 1}$, $j \in [K] := \{1, 2, \ldots, K\}$ and K is the total number of input vectors. Due to limited computing power, executing each task in a single UAV can be time consuming when the task size is large. To speed up the computation, the master node cooperates with its neighboring N UAVs within the communication range by sharing with them the computation loads. These neighboring UAVs, referred to as the *worker nodes*, execute tasks assigned by the master node and send back the obtained results. The master node then aggregates the results to output the final values.

This chapter seeks the fastest way for the aforementioned NAC system to complete all tasks. The desired features of the approach include: 1) Resilience to the various uncertain system disturbances prominent in the UAV network, such as communication bottlenecks in the network traffic, link/node failures, package losses, and slow-downs of computing nodes; 2) Adaptivity for unpredictable network changes such as random node movement, topology and resource (e.g., communication, computing, and energy) changes; and 3) quick deployment without requiring any knowledge of the system models.

4.4 Dynamic Batch-Processing Based Coded Computation Framework

In this section, we first introduce a dynamic batch-processing based coded computation (D-BPCC) framework to enable robust cooperative airborne computing in an uncertain airspace. Under this framework, we then formulate the system optimization problems mathematically for the two different NAC formation scenarios.

4.4.1 D-BPCC Framework

The D-BPCC framework (see Fig. 4.1) exploits the coding theory to enhance system resilience to uncertain system disturbances and uses a dynamic batch-processing based procedure (extended from our previous design for static networks [157]) to make the NAC system adaptable to unpredictable network changes. In particular, in each worker node $i \in [N] := \{1,2,\ldots,N\}$, we encode matrix $\mathbf{A}$ into a new matrix $\hat{\mathbf{A}}_i \in \mathbb{R}^{p\times m}$ using the following equation

$$\hat{\mathbf{A}}_i = \mathbf{G}_i\mathbf{A},$$

where $\mathbf{G}_i \in \mathbb{R}^{p\times p}$ is an encoding matrix satisfying the condition that any p rows of the concatenated encoding matrix $\mathbf{G} = [\mathbf{G}_1;\mathbf{G}_2;\ldots;\mathbf{G}_N] \in \mathbb{R}^{Np\times p}$ are linearly independent. This step is computed offline, and $\hat{\mathbf{A}}_i$ and $\mathbf{G}$ are pre-stored in each worker node i, assuming the storage space of each worker node is sufficiently large.

Once receiving an input vector $\mathbf{x}_j$, the master node sends $\mathbf{x}_j$ to each worker node i. At the same time, it also notifies each node i the number of rows $h_{i,j}$ of $\hat{\mathbf{A}}_i$ to be processed at a time. In another word, each worker node i will evenly divide $\hat{\mathbf{A}}_i$ row-wise into $\lceil \frac{p}{h_{i,j}} \rceil$ submatrices as $[\hat{\mathbf{A}}_{i,1},\ldots,\hat{\mathbf{A}}_{i,\lceil \frac{p}{h_{i,j}} \rceil}]$, where each submatrix has $h_{i,j}$ rows except the last one $\hat{\mathbf{A}}_{i,\lceil \frac{p}{h_{i,j}} \rceil}$ that has $p-(\lceil \frac{p}{h_{i,j}} \rceil - 1)h_{i,j}$ rows. Each submatrix will then be multiplied with $\mathbf{x}_j$ one by one, i.e., $\hat{A}_{i,k}\mathbf{x}_j, \forall k \in [\lceil \frac{p}{h_{i,j}} \rceil]$. For ease of reference, we hereinafter call each submatrix of $\hat{\mathbf{A}}_i$ as a *batch*, and $h_{i,j}$ as the *batch size*. Once a batch is processed, the worker node will send the result back to

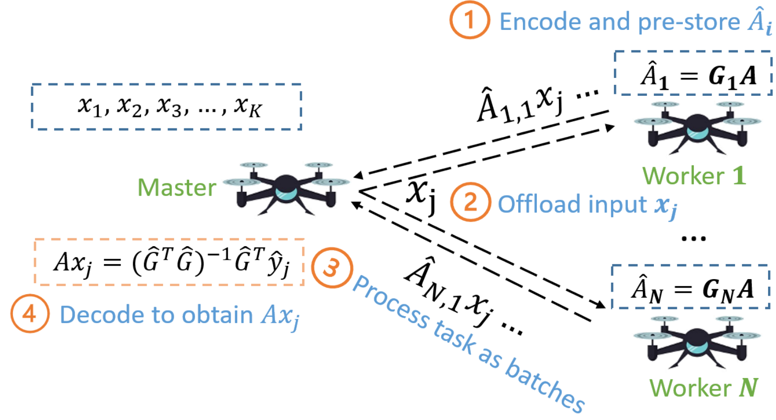


Figure 4.1. Cooperative airborne computing of matrix-vector multiplication tasks under the D-BPCC framework.

the master node immediately and move on to process the next batch.

The worker nodes will stop processing after receiving the notification from the master node, who will send such notification after it receives sufficient results for generating the output. In particular, let $\hat{\mathbf{y}}_j$ denote the results received at the master node by a certain time, which can be represented by

$$\hat{\mathbf{y}}_j = \hat{\mathbf{G}}\mathbf{A}\mathbf{x}_j,$$

where $\hat{\mathbf{G}}$ is a submatrix of $[\mathbf{G}_1; \mathbf{G}_2; \dots; \mathbf{G}_N]$. Then, the master node can generate the output using the following equation

$$\mathbf{A}\mathbf{x}_j = (\hat{\mathbf{G}}^T\hat{\mathbf{G}})^{-1}\hat{\mathbf{G}}^T\hat{\mathbf{y}}_j,$$

as long as the length of $\hat{\mathbf{y}}_j$ is larger than or equal to p, i.e., $|\hat{\mathbf{y}}_j| \geq p$.

The following lemma shows the resilience of D-BPCC to uncertain stragglers.

Lemma 8. *For a distributed computing system with a master node and N worker nodes, D-BPCC can tolerate up to $N-1$ worker nodes failures when processing matrix-vector multiplication tasks.*

Proof. As each worker node i pre-stores the encoded matrix $\hat{\mathbf{A}}_i = \mathbf{G}_i\mathbf{A}$ with $\hat{\mathbf{A}}_i \in \mathbb{R}^{p\times m}$, given an input $\mathbf{x} \in \mathbb{R}^{m\times 1}$, the aggregated results computed by the node are sufficient for obtaining the

value of $\mathbf{Ax}$ by $\mathbf{Ax} = \mathbf{G}_i^{-1}\hat{\mathbf{A}}_i\mathbf{x}$. Therefore, when there are $N-1$ or fewer worker nodes failing to return results, the master node can utilize the results from other nodes to recover the final value.

□

Remark 1. *According to Lemma 8, with D-BPCC, the master node can complete each computation task* $\mathbf{Ax}_j$ *as long as there is a functioning worker node. If the master node also shares certain workload (i.e., also being one of the worker nodes), any network changes or uncertain system disturbances won't cause tasks to fail as long as the master node is functional.*

Remark 2. *In the special case that* $N-1$ *worker nodes fail, the remaining functional worker node* i *will compute the whole task* $\hat{\mathbf{A}}_i\mathbf{x}$*. Although this may lead to more computation time compared with directly performing the task at the master node without distributed computing, the probability of this happening is usually small especially when* N *is large.*

The high resilience makes D-BPCC suitable for NAC in an uncertain and dynamic airspace. Processing tasks as small batches also naturally handles node heterogeneity as nodes with more computing or communication resources will process more batches. Moreover, with batch processing, the master node will continuously receive partial results from the worker nodes, which can be utilized to generate approximated outputs. This feature is crucial for safe UAV operations that require quick responses to environmental changes such as wind, birds, obstacles, and other UAVs.

4.4.2 Problem Formulation

In D-BPCC, the batch size $h_{i,j}$ is a key control variable to be determined, which will impact the system performance. In particular, when $h_{i,j}$ is large (e.g., equal to p), very few worker nodes will essentially contribute to the computation and their resources are hence underutilized. On the contrary, when $h_{i,j}$ is small (e.g., equal to 1), the frequent data transmissions by the worker nodes may generate large overhead and communication traffic. Another key factor that will impact the computing performance is the relative distance between two UAVs, which

affects the data transmission time. In the NAC formation scenario where UAVs move randomly with uncontrollable mobility, we exploit the optimization of the batch size $h_{i,j}$ to minimize the impact of random UAV mobility and other uncertain system disturbances, and make the system adaptable to uncertain network changes. In the scenario where UAVs are controllable, we exploit the benefit of UAV mobility control to computing and jointly optimize the batch size and UAV mobility.

To formulate the system optimization problems mathematically, we first introduce the evaluation metrics for the system performance. Denote the time required by each worker node i to process $b_{i,j}$ batches for the j-th task as $T_{i,j}$. Then $T_{i,j}$ can be captured by the following equation.

$$T_{i,j} = T_{i,j}^{comm} + T_{i,j}^{comp}, \tag{4.1}$$

which includes the communication time $T_{i,j}^{comm}$ and the computation time $T_{i,j}^{comp}$. The communication time $T_{i,j}^{comm}$ can be captured by

$$T_{i,j}^{comm} = T_{i,j}^{comm}(\mathbf{x}_j) + T_{i,j}^{comm}(\hat{A}_{i,b_{i,j}}\mathbf{x}_j)$$

where $T_i^{comm}(\mathbf{x}_j)$ is the time spent to send the input vector $\mathbf{x}_j$ from the master node to the worker node i and $T_{i,j}^{comm}(\hat{A}_{i,b_{i,j}}\mathbf{x}_j)$ is the time spent to send the last batch computation result from the worker node i back to the master node. Note that there is no break between two batches. Once a worker node completes a batch, it will immediately move on to process the next batch and at the same time transmit the computation result of the previous batch to the master node.

The computation time $T_{i,j}^{comp}$ in (4.1) can be captured by

$$T_{i,j}^{comp} = \sum_{k=1}^{b_{i,j}} T_{i,j}^{comp}(\hat{A}_{i,k}, \mathbf{x}_j)$$

where $T_{i,j}^{comp}(\hat{\mathbf{A}}_{i,k}, \mathbf{x}_j)$ is the time spent by the worker node i to compute one batch $\hat{A}_{i,k}\mathbf{x}_j$. Note that the communication and computation times are affected by various factors and finding perfect models for them is very challenging considering the uncertain and dynamic airspace. In contrast with most existing studies that assume the existence of perfect communication and computation models, we do not make such assumptions in this study.

With $T_{i,j}$, the completion time of each task $j \in [K]$ can then be represented by:

$$T_j = \min_t \{t | R_j(t) \geq p\}$$

where $R_j(t) = \sum_{i=1}^{N} b_{i,j} h_{i,j} \mathbb{1}_{T_{i,j} \leq t}$ is the total number of rows of inner product results for task j that the master node has received by time t. $\mathbb{1}$ is the indicator function [169].

The mathematical formulations of the optimization problems for the two NAC formation scenarios are described as follows.

Problem 1

Consider the scenario where the mobility of UAVs is random and cannot be controlled. Assuming that the positions and velocities of the UAVs can be observed through sensing and estimation, we aim to minimize the total task completion time by optimizing the batch size $h_{i,j}$, which is formulated as

$$\begin{aligned}
\mathscr{P}_1 : \min_{\substack{h_{i,j} \\ \forall i \in [N], \forall j \in [K]}} \quad & J_1 = \sum_{j=1}^{K} T_j \\
\text{s.t.} \quad & \mathscr{C}_1 : h_{i,j} \in \mathbb{Z}^+, \forall i \in [N], \forall j \in [K] \\
& \mathscr{C}_2 : h_{i,j} \leq p, \forall i \in [N], \forall j \in [K] \\
& \mathscr{C}_3 : \mathbf{p}_{i,t+\Delta T} = f_i(\mathbf{p}_{i,t}, \mathbf{v}_{i,t}, \Delta T), \forall i \in [N+1] \\
& \mathscr{C}_4 : T_j = \min_t \ \{t | R_j(t) \geq p\}, \forall j \in [K] \\
& \mathscr{C}_5 : R_j(t) = \sum_{i=1}^{N} b_{i,j} h_{i,j} \mathbb{1}_{T_{i,j} \leq t}, \forall j \in [K]
\end{aligned} \tag{4.2}$$

In constraint $\mathscr{C}_3$, $\mathbf{p}_{i,t}, \mathbf{v}_{i,t}$ denote the position and velocity of UAV i at time t, respectively, where $i \in [N+1]$ with the master node indexed by $N+1$. $f_i(\cdot)$ describes the movement behavior of each worker node i, whose specific formula is unknown, but the velocity of each UAV is assumed to be fixed over a small time period ΔT.

Problem 2

Consider the scenario where the mobility of UAVs is controllable, and each UAV has a target location to reach while performing cooperative airborne computing. The goal is to simultaneously minimize the total task completion time and the total UAV flight time. To achieve this goal, we formulate the following optimization problem that jointly optimizes UAVs' velocities and batch sizes.

$$
\begin{aligned}
\mathscr{P}_2: \min_{\substack{h_{i,j},\mathbf{v}_{i,j} \\ \forall i\in[N],\forall j\in[K]}} \quad & J_2 = \omega \sum_{j=1}^{K} T_j + \sum_{i=1}^{N+1} T_i^{flight} \\
\text{s.t.} \quad & \mathscr{C}_1, \mathscr{C}_2, \mathscr{C}_3, \mathscr{C}_4, \mathscr{C}_5 \\
& \mathscr{C}_6: \mathbf{v}_{i,t} = \mathbf{v}_{i,j}, t_j \le t < t_{j+1}, \forall i \in [N+1] \\
& \mathscr{C}_7: \mathbf{p}_{i,T_i^{flight}} = \mathbf{g}_i, \forall i \in [N+1] \\
& \mathscr{C}_8: |\mathbf{p}_{i,t} - \mathbf{p}_{j,t}| \le \varepsilon, \forall i,j \in [N+1], i \ne j \\
& \mathscr{C}_9: \mathbf{v}_{min} \le \mathbf{v}_{i,j} \le \mathbf{v}_{max}, \forall i \in [N+1], j \in [K]
\end{aligned}
\tag{4.3}
$$

where T_i^{flight} is the flight time of UAV i and ω is a weight that trades off between total task completion time and total flight time. When executing each task j, we let each UAV i fly at a velocity of $\mathbf{v}_{i,j}$, which remains unchanged during the task execution period $[t_j, t_{j+1})$, where t_j is the start time of task j and $t_{j+1} = t_j + T_j$. Constraint $\mathscr{C}_7$ ensures that each UAV will reach its target location specified by $\mathbf{g}_i$. Constraint $\mathscr{C}_8$ prevents collisions among the UAVs, where ε is the minimum safety distance between two UAVs.

4.5 DRL-based Solution to $\mathscr{P}_1$

In this section, we solve problem $\mathscr{P}_1$ in (4.2) by exploiting DRL, specifically, the Deep Deterministic Policy Gradient (DDPG) method [170], which does not need any knowledge of the communication, computation or UAV mobility models, and can be quickly deployed in any NAC systems to make decisions online in real time. We first convert the optimization problem into a reinforcement learning (RL) problem, and then describe the solution to the resulting problem.

4.5.1 RL based Formulation for $\mathscr{P}_1$

To convert $\mathscr{P}_1$ into a RL problem, we first model this problem as a Markov Decision Process (MDP) characterized by a quintuple $(\mathbf{s}_t, \mathbf{a}_t, \zeta, r, \mu)$ with each component described as follows.

State $\mathbf{s}_t$

The system state at time t includes distances $d_{i,t}$ between each worker node and the master node, and velocities $\mathbf{v}_{i,t}$ of all nodes. Specifically,

$$\mathbf{s}_t = [d_{1,t}, d_{2,t}, \dots, d_{N,t}, \mathbf{v}_{1,t}, \mathbf{v}_{2,t}, \dots, \mathbf{v}_{N+1,t}]^\top \in \mathscr{S},$$

where $\mathscr{S}$ denotes the state space.

Action $\mathbf{a}_t$

In $\mathscr{P}_1$, the batch size $h_{i,j}$ for each worker node i and task j is the control variable to be determined when executing the task j, hence the action to take. The action taken at time t_j (start time of task j) is then defined as follows

$$\mathbf{a}_{t_j} = [h_{1,j}, h_{2,j}, h_{3,j}, \dots, h_{N,j}]^\top \in \mathscr{A}$$

where $\mathscr{A}$ is the action space. Note that each $h_{i,j}$ in $\mathbf{a}_{t_j}$ should satisfy constraints $\mathscr{C}_1$ and $\mathscr{C}_2$ in $\mathscr{P}_1$, i.e., $h_{i,j} \in \mathbb{Z}^+$ and $h_{i,j} \leq p$.

Transition function ζ

The transition function describes the transition from the current state to the next state given the current action, and outputs the probability distribution over the next state, i.e., ζ : $\mathscr{S} \times \mathscr{A} \times \mathscr{S} \to [0,1]$. In $\mathscr{P}_1$, given the state $\mathbf{s}_{t_j}$ and action $\mathbf{a}_{t_j}$ at time t_j, the next state of interest is the state $\mathbf{s}_{t_{j+1}}$ at the start time t_{j+1} of the next task $j+1$, as t_{j+1} is the time to make the next decision. To obtain $\mathbf{s}_{t_{j+1}}$, the computing nodes take action $\mathbf{a}_{t_j}$ and execute task j. The next state $\mathbf{s}_{t_{j+1}}$ can then be obtained by observing the positions and velocities of all nodes at time $t_{j+1} = t_j + T_j$. Note that in our settings, the explicit form of the transition function ζ is unknown.

Reward function r

As the goal of $\mathscr{P}_1$ is to minimize the total task completion time, we define the reward function as follows:

$$r(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}) = -\omega_1 T_j$$

which can be obtained based on the state $\mathbf{s}_{t_j}$ and action $\mathbf{a}_{t_j}$. A larger reward indicates less time taken for computing a task j.

Policy function μ

The policy function determines the action to take given the current state, which can be deterministic or stochastic. We here consider a deterministic policy function that maps the state space to the action space, i.e., $\mu : \mathscr{S} \to \mathscr{A}$ and $\mathbf{a} = \mu(\mathbf{s})$. In $\mathscr{P}_1$, the policy function is called only at the start time t_j of each task j.

With the above MDP setting, we can then convert $\mathscr{P}_1$ into a RL problem that determines the optimal policy $\mu^*(\mathbf{s})$ such that the following expected cumulative discounted reward is

maximized,

$$Q^{\mu}(\mathbf{s},\mathbf{a}) := \mathbb{E}_{\substack{\mathbf{s}_{t_j}\sim\zeta \\ \mathbf{a}_{t_j}=\mu(\mathbf{s}_{t_j})}} \left[\sum_{j=1}^{K} \gamma^{j-1} r(\mathbf{s}_{t_j},\mathbf{a}_{t_j}) | \mathbf{s}_{t_1}=\mathbf{s}, \mathbf{a}_{t_1}=\mathbf{a}\right]$$

where $\mathbf{s}\in\mathscr{S}$, $\mathbf{a}\in\mathscr{A}$, and $\gamma\in(0,1]$ is a discount factor. $Q^{\mu}(\mathbf{s},\mathbf{a})$ is also known as the action value function or Q function. The optimal policy can be obtained by

$$\mu^*(\mathbf{s}) \in \arg\max_{\mathbf{a}} Q^*(\mathbf{s},\mathbf{a})$$

where $Q^*(\mathbf{s},\mathbf{a})$ is the optimal Q function obtained by $Q^*(\mathbf{s},\mathbf{a}) := \max_{\mu} Q^{\mu}(\mathbf{s},\mathbf{a})$.

4.5.2 Deterministic Policy Gradient Method

To solve the above RL problem, the key is to derive the optimal Q function $Q^*(\mathbf{s},\mathbf{a})$, which can be obtained by using the Bellman equation [171] as follows

$$Q^*(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathbf{s}'\sim\zeta}\left[r(\mathbf{s},\mathbf{a}) + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}',\mathbf{a}')\right] \tag{4.4}$$

where $\mathbf{s}'\sim\zeta(\mathbf{s},\mathbf{a})$ denotes the next state. As the state space is continuous, $Q^*(\mathbf{s},\mathbf{a})$ cannot be directly computed. Instead, we approximate $Q^*(\mathbf{s},\mathbf{a})$ using a parameterized non-linear function, denoted as $Q(\mathbf{s},\mathbf{a};\theta)$, where θ is the parameter. We delay the design of the non-linear function to the next subsection.

It is noted that in (4.4), the transition function ζ is needed for computing the optimal Q function, whose explicit form is, however, unknown. To address this challenge, we introduce a learning *agent* to interact with the NAC system, which is the *environment*, and collect transition data to approximate the transition function ζ. The transition data to be collected includes the state, action, reward, and the next state, i.e., $(\mathbf{s},\mathbf{a},r,\mathbf{s}')$, and are stored in a *replay buffer* denoted

by $\mathscr{D}$. Equation (4.4) can then be rewritten as

$$Q(\mathbf{s},\mathbf{a};\theta) \approx \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathscr{D}}\left[r(\mathbf{s},\mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}',\mathbf{a}';\theta)\right],$$

with $Q^*(\mathbf{s},\mathbf{a})$ approximated by $Q(\mathbf{s},\mathbf{a};\theta)$. This equation can be solved to obtain the optimal Q function by minimizing the following error,

$$J(\theta) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathscr{D}}\left[\left(Q(\mathbf{s},\mathbf{a};\theta) - \left(r + \gamma \max_{\mathbf{a}'} Q\left(\mathbf{s}',\mathbf{a}';\theta\right)\right)\right)^2\right].$$

However, we note that directly computing $\max_{\mathbf{a}'} Q(\mathbf{s}',\mathbf{a}';\theta)$ in the above error function is difficult considering the large state and action spaces. To address this issue, we introduce a policy approximator denoted by $\mu(\mathbf{s};\phi)$ with parameter ϕ to output the action using $\mathbf{a} = \mu(\mathbf{s};\phi)$. We then learn the parameters of the policy and Q functions to minimize the error function. Moreover, to stabilize the training procedure, we introduce a target policy function μ' and a target Q function Q' with the same function representations and initial weights as the original policy and Q functions. The error function finally used for finding the optimal Q function is then given as

$$J(\theta) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathscr{D}}\left[\left(Q(\mathbf{s},\mathbf{a};\theta) - \left(r + \gamma Q'\left(\mathbf{s}',\mu'(\mathbf{s}';\phi);\theta\right)\right)\right)^2\right]$$

where the parameters ϕ of the policy function is updated using the policy gradient theorem by

$$\nabla_\phi J(\phi) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathscr{D}}\left[\nabla_\phi \mu(\mathbf{s};\phi) \nabla_{\mathbf{a}} Q(\mathbf{s},\mathbf{a};\theta)\right]$$

4.5.3 Deep Neural Network based Function Representation

To approximate the policy functions, $\mu(\mathbf{s})$ and $\mu'(\mathbf{s})$, and the Q functions, $Q(\mathbf{s},\mathbf{a})$ and $Q'(\mathbf{s},\mathbf{a})$, we adopt deep neural networks (DNNs) considering their powerful approximation

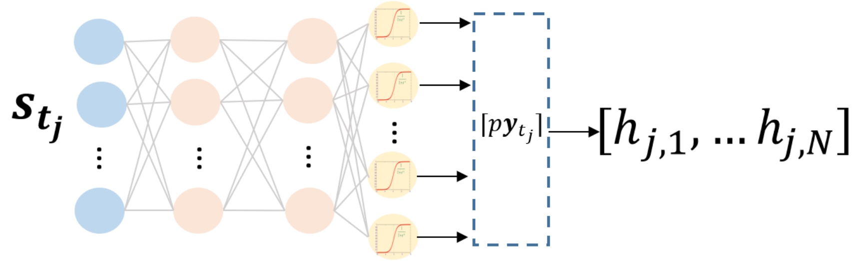


Figure 4.2. DNN representation of the policy functions for computation load optimization.

capability. For the policy functions, we design a DNN with four fully connected layers. The input layer consists of $3N+2$ units representing the system state at time t_j (start time of task $j \in [K]$), denoted by $\mathbf{s}_{t_j}$. The output layer consists of $N+1$ sigmoid units [172], denoted by $\mathbf{y}_{t_j}$, which output normalized batch sizes ranged between 0 and 1. The batch size $h_{i,j}$ for each worker i and task j is computed by

$$h_{i,j} = \lceil p\mathbf{y}_{i,t_j} \rceil$$

where $\lceil \rceil$ is the ceiling function. This ensures that the load constraints $\mathscr{C}_1, \mathscr{C}_2$ are satisfied. Each hidden layer consists of 64 units with ReLU activation function [173]. An illustration of the designed DNN is shown in Fig. 4.2.

To approximate the Q functions, we design a DNN also with four fully connected layers. It takes both state $\mathbf{s}_{t_j}$ and action $\mathbf{a}_{t_j}$ as the input and has a single linear unit in the output layer to generate the Q value. The hidden layers consist of 64 ReLU units.

4.5.4 Training DRL Agent

To estimate the parameters in the DNN-based policy and Q functions, we adopt the offline training procedure in [170], which involves the following three stages.

Initialization

In the initialization stage (Lines 1-6 in Alg. 2), the weights of the DNNs are randomly initialized. The learning agent then executes the policy constructed using the initial weights for $initial_episode_number$ episodes to initialize the replay buffer $\mathscr{D}$. After that, the exploration and

parameter update stages described below are performed for *max_training_iteration* iterations, where *max_training_iteration* is a constant large enough for ensuring convergence.

Training Data Collection

In the training data collection stage (Lines 8-12 in Alg. 2), the learning agent interacts with the environment to collect the transition data $(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}, r, \mathbf{s}_{t_{j+1}})$ by running the policy constructed with the current weights for *max_episode_number* episodes. In particular, in time step t_j for each task j, given the current state $\mathbf{s}_{t_j}$, the action is generated by applying the policy $\mathbf{a}_{t_j} = \mu(\mathbf{s}_{t_j})$. The next state $\mathbf{s}_{t_{j+1}}$ is obtained by observing the positions $\mathbf{p}_{i,t_{j+1}}$ and velocities $\mathbf{v}_{i,t_{j+1}}$ of all UAVs at time t_{j+1}. The movement of each UAV is described by the mobility model $\mathbf{p}_{i,t_{j+\Delta T}} = f_i(\mathbf{p}_{i,t}, \mathbf{v}_{i,t}, \Delta T)$ with $\mathbf{v}_{i,t_j}$ being the mobility control input. It is noted that our method does not require knowledge of the mobility model and f_i can take any form. With $\mathbf{p}_{i,t_{j+1}}$, the distances $d_{i,t_{j+1}}$ in the next state $\mathbf{s}_{t_{j+1}}$ can then be computed. The rewards r are obtained by applying the reward function described in Sec. 4.5.1.

The collected transition data is stored in the replay buffer $\mathscr{D}$. After that, a mini-batch $\mathscr{B}$ is randomly sampled from the replay buffer $\mathscr{D}$, which will be used in the next stage to update the parameters of the policy and Q functions.

Parameter Update

The parameters θ of the Q function are updated by minimizing the following temporal-difference error:

$$J(\theta) \approx \frac{1}{|\mathscr{B}|} \sum_{(\mathbf{s},\mathbf{a},\mathbf{s}',r)\in\mathscr{B}} \left[L(\mathbf{s}',r) - Q(\mathbf{s},\mathbf{a};\theta)\right]^2$$
$$L(\mathbf{s}',r) = r + \gamma Q'(\mathbf{s}', \mu'(\mathbf{s}';\phi);\theta) \tag{4.5}$$

The policy parameters ϕ are updated using gradient ascent based on the policy gradient theorem, with the gradient given as follows[171]:

$$\nabla_\phi J(\phi) \approx \frac{1}{|\mathscr{B}|} \sum_{(\mathbf{s},\mathbf{a},\mathbf{s}',\mathbf{r})\in\mathscr{B}} \nabla_\phi \mu(\mathbf{s};\phi) \nabla_a Q(\mathbf{s},\mathbf{a};\theta) \tag{4.6}$$

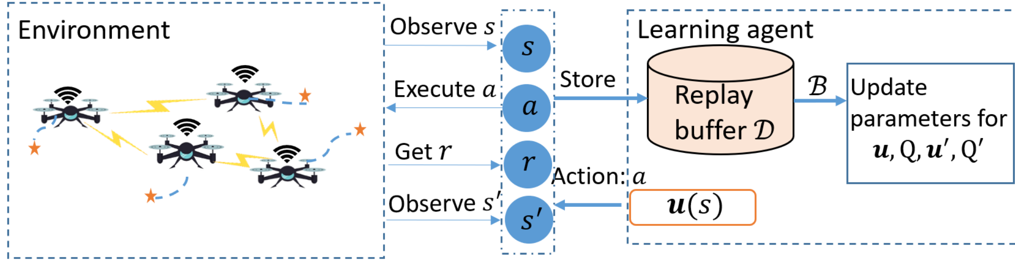


Figure 4.3. The training process of the DRL-based method for NAC with uncontrollable UAVs.

To update the parameters, ϕ' and θ', in the target policy and Q functions, Polyak averaging given below is applied:

$$\begin{aligned} \phi' &\leftarrow \tau\phi' + (1-\tau)\phi \\ \theta' &\leftarrow \tau\theta' + (1-\tau)\theta \end{aligned} \tag{4.7}$$

where $\tau \in (0,1)$ is a hyperparameter. The complete training procedure is summarized in Alg. 2 and illustrated in Fig. 4.3.

Algorithm 2: DRL Training Procedure

```
   // Initialization
1  Initialize φ, θ, θ', φ', 𝒟
2  for k = 1 : initial_episode_number do
3      for j = 1 : K do
4          Select a_{t_j} = μ(s_{t_j}; φ).
5          Execute action a_{t_j} and receive new state s_{t_{j+1}} and reward r.
6          Store (s_{t_j}, a_{t_j}, r, s_{t_{j+1}}) in replay buffer 𝒟.
7  for iteration = 1 : max_training_iteration do
       // Training Data Collection
8      for k = 1 : max_episode_number do
9          for j = 1 : K do
10             Select a_{t_j} = μ(s_{t_j}; φ).
11             Execute action a_{t_j} and receive new state s_{t_{j+1}} and reward r.
12             Store (s_{t_j}, a_{t_j}, r, s_{t_{j+1}}) in replay buffer 𝒟.
13     Sample a random mini-batch ℬ.
       // Parameter Update
14     Update θ by minimizing the temporal-difference error in (4.5).
15     Update φ using gradient ascent, with the gradient provided in (4.6).
16     Update θ and φ using (4.7).
```

4.5.5 Convergence and Complexity Analysis

Our DRL-based algorithm follows the standard DDPG training procedure, which is not theoretically guaranteed to converge in its general form [174]. In this study, we evaluate the convergence of the proposed algorithm empirically through simulation studies in Section 6.5.

The time complexity of the proposed DRL-based algorithm is dominated by the training of actor and critic neural networks [175, 176], which is captured by $O(\sum_{j=1}^{J-1} \chi_j \chi_{j+1} + \sum_{z=1}^{Z-1} \hat{\chi}_z \hat{\chi}_{z+1})$. Here J and Z are the number of layers in the actor and critic neural networks, respectively. χ_j, and $\hat{\chi}_z$ represent the number of units in j-th and z-th layer, respectively.

The space complexity of our algorithm is determined by the amount of memory required to store the actor and critic neural networks as well as the replay buffer [175], which is captured by $O(\sum_{j=1}^{J-1} \chi_j \chi_{j+1} + \sum_{z=1}^{Z-1} \hat{\chi}_z \hat{\chi}_{z+1} + |\mathscr{D}|)$.

In our design, $J = 4, Z = 4$, $\chi_1 = \dim(\mathbf{s}), \chi_2 = \chi_3 = 64, \chi_4 = \dim(\mathbf{a}), \hat{\chi}_1 = \dim(\mathbf{s}) + \dim(\mathbf{a}), \hat{\chi}_2 = \hat{\chi}_3 = 64, \hat{\chi}_4 = 1$, and $|\mathscr{D}| = 10^5$, where $\dim(\cdot)$ finds the dimension of a vector.

4.6 DRL-based Solution to $\mathscr{P}_2$

In this section, we solve problem $\mathscr{P}_2$ in (4.3) by extending the DRL method described in the previous section.

4.6.1 RL based Formulation for $\mathscr{P}_2$

Similarly, we first model problem $\mathscr{P}_2$ as a MDP characterized by the following quintuple $(\mathbf{s}_t, \mathbf{a}_t, \zeta, r, \mu)$.

State $\mathbf{s}_t$

As UAVs' velocities are control variables in $\mathscr{P}_2$, we define the state at time t to only include distances $d_{i,t}$ between each worker node i and the master node. The state $\mathbf{s}_t$ is then represented by

$$\mathbf{s}_t = [d_{1,t}, d_{2,t}, \dots, d_{N,t}]^\top \in \mathscr{S}$$

where $\mathscr{S}$ is the state space.

Action $\mathbf{a}_t$

In this setting, in addition to the batch size $h_{i,j}$ for each worker node i and task j, the action $\mathbf{a}_{t_j}$ taken at time t_j (start time of task j) also includes the velocities of each node $\mathbf{v}_{i,j}$, i.e.,

$$\mathbf{a}_{t_j} = [h_{1,j}, \ldots, h_{N,j}, \mathbf{v}_{1,j}, \ldots, \mathbf{v}_{N+1,j}]^\top \in \mathscr{A}$$

Transition function ζ

The explicit form of the transition function ζ is still unknown in $\mathscr{P}_2$. Different from $\mathscr{P}_1$, in order to obtain the next state $\mathbf{s}_{t_{j+1}}$, we should let the NAC system execute task j and UAVs move at the same time based on the action $\mathbf{a}_{t_j}$. $\mathbf{s}_{t_{j+1}}$ can then be obtained by observing the positions of the UAVs at time $t_{j+1} = t_j + T_j$.

Reward function r

$\mathscr{P}_2$ aims to minimize the weighted sum of the total task completion time $\sum_{j=1}^{K} T_j$ and the total flight time $\sum_{i=1}^{N+1} T_i^{flight}$. To achieve this goal, given current state $\mathbf{s}_{t_j}$ and action $\mathbf{a}_{t_j}$, we define the reward function as follows

$$r(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}) = -\omega_1 T_j - \sum_{i=1}^{N+1} ||\mathbf{p}_{i,t_j} - \mathbf{g}_i||$$

where the first term penalizes long task completion time, and the second term drives each UAV to move closer to its target location and hence arrive there sooner. Moreover, as UAVs should keep a safe distance between each other (constraint $\mathscr{C}_8$ in $\mathscr{P}_2$), we add a third term into the reward function to achieve collision avoidance. The revised reward function is then given by

$$r(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}) = -\omega_1 T_j - \sum_{i=1}^{N+1} ||\mathbf{p}_{i,t_j} - \mathbf{g}_i|| - \omega_c \mathbb{1}_{|\mathbf{p}_{i,t} - \mathbf{p}_{k,t}| \leq \varepsilon, \forall i,k \in [N+1], i \neq k} \tag{4.8}$$

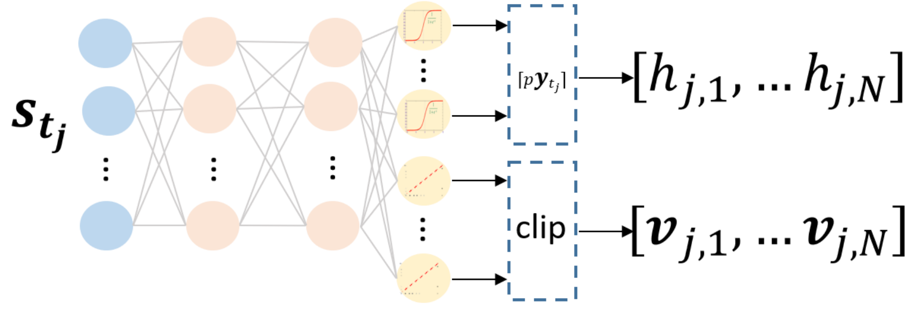


Figure 4.4. DNN representation of policy function μ for joint computation load and UAV mobility optimization.

where $\omega_c > 0$ is the weight.

Policy function μ

Similar as $\mathscr{P}_1$, a deterministic policy function $\mu : \mathscr{S} \rightarrow \mathscr{A}$ is considered and used to generate the action at the start time t_j of each task $j \in [K]$.

With the above MDP setting, we can then follow the similar procedure described in Sec. 4.5.1 to formulate the RL problem.

4.6.2 Solution to $\mathscr{P}_2$

The derived RL problem can be solved by using the deterministic policy gradient method described in Sec. 4.5.2. For function approximation, we adopt the same DNN structure to approximate the Q functions. To approximate the policy functions, we design a DNN shown in Fig. 4.4. It differs from the one shown in Fig. 4.2 in that the output layer contains additional $2N+2$ linear units for generating UAV mobility control signals, i.e., $\mathbf{v}_{i,j} \in \mathbb{R}^2, \forall i \in [N+1]$. To meet constraint $\mathscr{C}_9$ in $\mathscr{P}_2$, the values generated by these linear units are clipped if falling out of the range $[\mathbf{v}_{min}, \mathbf{v}_{max}]$.

To ensure each UAV will reach its target location (constraint $\mathscr{C}_6$ in $\mathscr{P}_2$), which can happen before or after it completes all computation tasks, we introduce the following mechanism. In case when the UAV has completed all assigned tasks but hasn't reached its target location yet, the UAV switches to another policy that generates mobility control commands only based on its

current state. This policy is trained using a similar DRL method with the reward function defined as

$$r(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}) = -\sum_{i=1}^{N+1} ||\mathbf{p}_{i,t_j} - \mathbf{g}_i|| - \omega_c \mathbb{1}_{|\mathbf{p}_{i',t} - \mathbf{p}_{j',t}| \leq \varepsilon, \forall i', j' \in [N+1], i' \neq j'}.$$

The DNN used for approximating the policy function is similar to the one shown in Fig. 4.2 but with only $2N+2$ linear units for generating UAV velocities. Moreover, the UAV changes its velocity after every ΔT. In other cases when the UAV has reached its target location but still has computation tasks remain to complete, the UAV switches to the policy trained using the method presented in Sec. 4.5 for generating the task allocation decisions.

4.7 Simulation Studies

In this section, we conduct simulation studies to evaluate the performance of the proposed DRL and D-BPCC based methods for NAC under two different formation scenarios. We first design a simulator for the NAC system, and then describe the benchmark schemes used in the comparative studies. The experiment results are presented at the end. All experiments were run on an Alienware Desktop with 32GB memory, 16-cores CPU with 3.6GHz.

4.7.1 Simulator Design

To simulate the NAC system, we adopt the following communication, computation and mobility models. It is worth noting that our methods are general and can be applied to NAC systems described by other system models, as no knowledge of the models are required to implement our methods.

Communication Model

Suppose all UAVs in the NAC system are equipped with the same directional antennas for long-range and broadband UAV-to-UAV communications, and implement advanced antenna

control algorithms to keep the antennas aligned for robust communication [177]. The time to transmit a matrix $\mathbf{X}$ with a rows and b columns between any two UAVs via the UAV-to-UAV link can then be modeled as

$$T^{comm}(\mathbf{X}) = \frac{a \times b \times u}{C}$$

where u (bits) is the average size of the elements in $\mathbf{X}$ and is set as 32 in all experiments. C (bits/sec) is the data rate that can be derived using the Shannon's theory as follows

$$C = W log_2(1 + \frac{S}{N_0})$$
$$S = 10^{\frac{(S_d - 30)}{10}}$$
$$S_d = P_t + 20\log_{10}(\lambda) - 20\log_{10}(4\pi) - 20\log_{10}(d) + G + \kappa$$

where W (Hz) is the communication bandwidth between two UAVs. N_0 (Watts) is the noise power. S (Watts) is the signal power determined by the transmitting power of the transmitter P_t (dBm), wave length λ (m), sum of the transmitting and receiving gains G (dBi), and distance between the two UAVs d (m). κ denotes the Gaussian noise with zero mean and variance σ [177]. In our simulations, these parameters are configured as $W = 10^4, N_0 = 1.1 \times 10^{-12}, P_t = 27, G = 32, \lambda = 0.12$ and $\sigma = 1$.

Computation Model

To simulate the computing power of a UAV, we extend the modeling technique used in many studies [114, 24]. Particularly, we assume the time $T_i^{comp}(\mathbf{A}, \mathbf{x})$ taken by each UAV i to multiply $\mathbf{A} \in \mathbb{R}^{\ell \times m}$ by $\mathbf{x} \in \mathbb{R}^{m \times 1}$ follows a shifted exponential distribution:

$$\mathrm{P}\left[T_i^{comp}(\mathbf{A}, \mathbf{x}) \leq t\right] = 1 - e^{-\frac{\beta_i}{\ell}(t - \alpha_i \ell - \xi_i)} \tag{4.9}$$

where $t \geq \alpha_i \ell + \xi_i$ specifies the minimum time required to compute the task.

$\beta_i > 0$ and $\alpha_i > 0$ are straggling and shift parameters, respectively, characterizing the

computing capability of the UAV. The bias term ξ_i captures the time required for task initialization and function calls. Of note, this term is not included in existing computation models. However, our experiments show that the shifted exponential model with a bias term better captures the characteristics of real computing systems. For illustration purpose, we plot in Fig. 4.5 the computation model constructed for the Amazon EC2 *t2.xlarge* instance by using real experiment data and following the parameter estimation procedure described in [157]. The estimated parameters are $\alpha = 6 \times 10^{-4}, \beta = 1265$ and $\xi = 0.04$.

In the following simulation studies, we let $\xi_i = \xi = 0.04$, $\forall i \in [N]$. The straggling parameter β_i is randomly generated from the range $[100, 500]$ and the shift parameter is set to $\alpha_i = \frac{1}{\beta_i}$ [156].

Mobility Model

We assume the UAVs are equipped with an advanced controller robust to wind perturbations and no strong winds are present. The point-mass kinematic model can then be used to simulate the movements of UAVs. In particular, the position of UAV i at time $t + \Delta T$ is computed by the following equation,

$$\mathbf{p}_{i,t+\Delta T} = f(\mathbf{p}_{i,t}, \mathbf{v}_{i,t}, \Delta T) = \mathbf{p}_{i,t} + \mathbf{v}_{i,t} \Delta T$$

In scenarios where the NAC system is formed by uncontrollable UAVs, we let each UAV i change its velocity after each computation task j is completed, with the velocity randomly picked from the range $[-10m/s, 10m/s] \times [-10m/s, 10m/s]$.

With the NAC simulator, we train the proposed DRL methods offline by following the procedure described in Algorithm 2. During the mission, the trained policies generate desired actions online in real time.

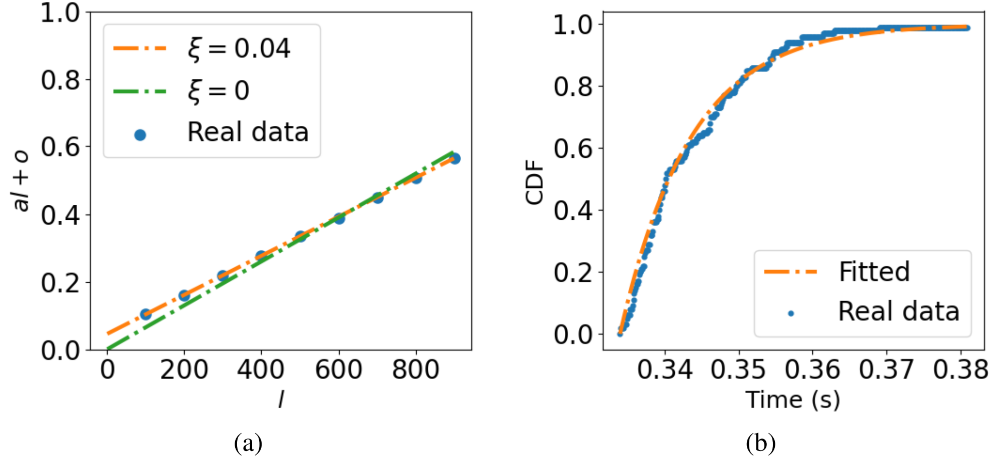


Figure 4.5. a) Minimum task completion time $\alpha l + \xi$ versus task size l. b) CDF of the task completion time of an Amazon EC2 *t*2.*xlarge* instance for computing $\mathbf{Ax}$ with $l = 500$.

4.7.2 Benchmarks

We implement the following four representative distributed computing schemes as benchmarks.

Uniform Uncoded (UU)

In the traditional uncoded distributed computing systems, to perform a matrix-vector multiplication task $\mathbf{Ax}$, the master node decomposes $\mathbf{A} \in \mathbb{R}^{p \times m}$ row-wise into N non-overlapping submatrices $\{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N\}$, where $\mathbf{A}_i \in \mathbb{R}^{\ell_i \times m}$, and assigns subtask $\mathbf{A}_i\mathbf{x}$ to work node $i \in [N]$. After receiving results from all worker nodes, the master node can recover $\mathbf{Ax}$ by concatenating the results, i.e., $\mathbf{Ax} = [\mathbf{A}_1\mathbf{x}; \mathbf{A}_2\mathbf{x}; \ldots; \mathbf{A}_N\mathbf{x}]$. To allocate the workload, the UU scheme [114] simply divides the load equally, i.e.,

$$\ell_i = \frac{p}{N}, \forall i \in [N],$$

disregarding the computing power of the worker nodes.

Load-Balanced Uncoded (LBU)

This scheme [156] divides the computation load according to the computing power of the worker nodes. In particular, the load assigned to each worker node i is inversely proportional to the expected time for this node to compute an inner product, i.e.,

$$\ell_i \propto \frac{\beta_i}{\alpha_i \beta_i + 1}, \forall i \in [N]$$

with $\sum_{i=1}^{N} \ell_i = p$. Note that this scheme requires the knowledge of the computation model.

Heterogeneous Coded Matrix Multiplication (HCMM)

This is a state-of-the-art CDC scheme for heterogeneous static computing systems [156]. It first encodes matrix $\mathbf{A}$ into a larger matrix $\hat{\mathbf{A}}$ with more rows, and then follows the same procedure as the uncoded schemes to partition and allocate the computation load. The only difference is that the submatrices of the encoded matrix $\hat{\mathbf{A}}$ are multiplied with the input vector at the worker nodes. When the master node receives sufficient results, i.e., the number of rows of aggregated results is no less than p, it can compute the final value. In this scheme, the load assigned to each worker node i is computed by

$$\ell_i = \frac{p}{\lambda_i \eta},$$

where λ_i is the positive solution to $e^{\beta_i \lambda_i} = e^{\alpha_i \beta_i}(\beta_i \lambda_i + 1)$, and $\eta = \sum_{i=1}^{N} \frac{\beta_i}{1+\beta_i \lambda_i}$. Like LBU, HCMM also requires the knowledge of the computation model.

Coded Cooperative Computation Protocol (C3P)

C3P [126] is a state-of-the-art CDC scheme for heterogeneous mobile computing systems. In this scheme, the master node packetizes each row of $\mathbf{A}$ and encodes each packet. Given an input vector $\mathbf{x}$, it first broadcasts $\mathbf{x}$ to all worker nodes and then gradually offloads the coded packets to the worker nodes one by one. To optimize the computing performance, the offloading

interval is dynamically adjusted based on the worker nodes' response times to previous tasks. This scheme does not require any knowledge of the computation, communication or mobility model, and hence can be directly used to solve problem $\mathscr{P}_1$.

As all benchmarks do not consider mobility control, to solve $\mathscr{P}_2$, we apply the benchmarks for load allocation and the DRL method described in Sec. 4.6.2 for UAV mobility control, which runs independently.

4.7.3 Evaluation of Solution to $\mathscr{P}_1$

This section evaluates our solution to $\mathscr{P}_1$ for the scenario where the mobility of UAVs is uncontrollable.

Experiment Settings

We consider the following three computation scenarios with varying number of UAVs and task sizes.

- **Scenario 1:** $N = 3$, $p = 5000$.
- **Scenario 2:** $N = 6$, $p = 10000$.
- **Scenario 3:** $N = 12$, $p = 20000$.

In all computation scenarios, the dimension of each input vector is set to $m = 10^5$. Initially, the UAVs are randomly distributed over a 400m $\times$ 400m area. The total number of computation tasks to be computed is $K = 25$ and the travel interval is set to $\Delta T = 10s$. To understand the impact of the bias term ξ in the computation model, we also evaluate the case when $\xi = 0$, in addition to the more realistic case when $\xi = 0.04$. In all experiments, the parameters of the DRL method are configured as $\gamma = 0.95, \tau = 0.01, \omega_1 = 15, \omega_c = 5$, $initial_episode_num = 205$, $max_training_iteration = 1000$, and $max_episode_num = 4$.

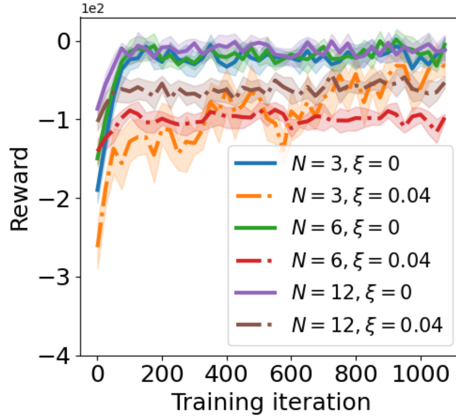


Figure 4.6. Training reward of our method for $\mathscr{P}_1$.

Training Reward

We first show the learning curves of our method in different computation scenarios with different bias settings. As shown in Fig. 4.6, our method converges in all scenarios.

Comparative Results

The first comparative study evaluates the computation efficiency of different methods. For each computation scenario, we run each method 100 times and record the mean time spent for completing each computation task, referred to as the *average task completion time*. As shown in Fig. 4.7, our method achieves the highest efficiency in all computation scenarios. Comparing Fig. 4.7(a) and Fig. 4.7(b), we can observe that the efficiency of both our method and C3P is significantly impacted by the value of ξ, the overhead induced by task initialization and function calls. However, ξ has a negligible impact on the performance of HCMM, LBU and UU. This is because the worker nodes in our method and C3P process each task batch by batch, where each packet in C3P can be considered as a batch with size $h_{i,j} = 1, \forall i \in [N], \forall j \in [K]$. Nevertheless, in HCMM, LBU, and UU, worker nodes process each task as a whole. Hence, when ξ is non-zero, the overhead induced by the many batches in our method and C3P can be significant.

Moreover, we can observe from Fig. 4.7(a) that although the efficiency of C3P is comparable to our method when the batch overhead is negligible, it is much slower than our method when the batch overhead cannot be ignored (see Fig. 4.7(b)). This is because the C3P fixes the

batch size to 1 and hence does not address the performance-cost trade-offs. Furthermore, as the C3P applies a simple moving average algorithm [126] to approximate the worker nodes' computation times when making the offloading decisions, it achieves a poor performance when the computation times have a large variance, which can happen if UAVs are conducting many other tasks at the same time. To illustrate this, we let the computing parameters μ and α of each worker node change frequently over time, by randomly sampling a new value from the range $[100, 500]$ after each batch is processed. The results are shown in Fig. 4.8. By comparing Fig. 4.7(a) and Fig. 4.8, we can observe that though the performance of both C3P and our method degrades as nodes' computing resources change frequently, our method is much more resilient to such changes. Of interest, UU is not impacted by such changes. This is because UU divides the load equally, disregarding the different computing capabilities of the worker nodes.

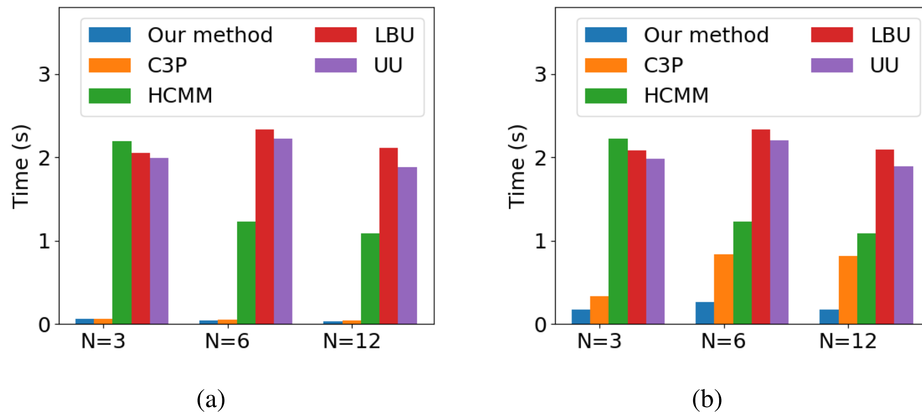


Figure 4.7. Average task completion times of different methods in different scenarios when a) $\xi = 0$ and b) $\xi = 0.04$.

The second comparative study evaluates the resilience of different methods to network topology changes caused by high UAV mobility. Particularly, we randomly pick one or two worker nodes and make them move out of the communication range of the master node, which is set to $1500m$. Therefore, results computed by the nodes left cannot be received by the master node.

Fig. 4.9 shows the simulation results with $\xi = 0.04$. As we can see, our method still achieves the highest efficiency and is the most resilient to topology changes. Of note, there

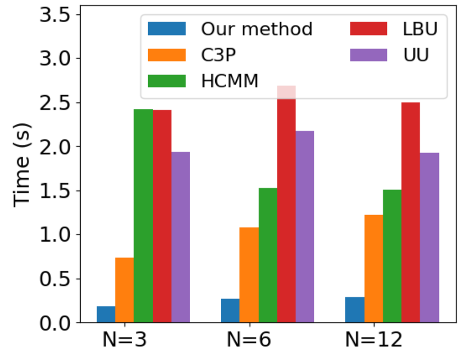


Figure 4.8. Average task completion times of different methods in different scenarios when $\xi = 0$ and computation times have a large variance.

is no data for the LBU and UU schemes, as they require results from all the worker nodes to successfully complete a computation task and hence any node leaving would cause the whole task to fail. To further evaluate the resilience of different methods, we perform a stress test and measure the *success rate* (ratio of successful runs) of each method when the number of nodes left increases. Fig. 4.10 shows the results for computation Scenario 2 with $N = 6$. The results for the other two scenarios are similar and thus are eliminated to save space. From the figure, we can see that both our method and C3P can complete all computation tasks as long as there is a worker node within the network. The success rate of HCMM decreases quickly when more nodes leave the network, and both UU and LBU fail all tasks when there is one or more nodes left.

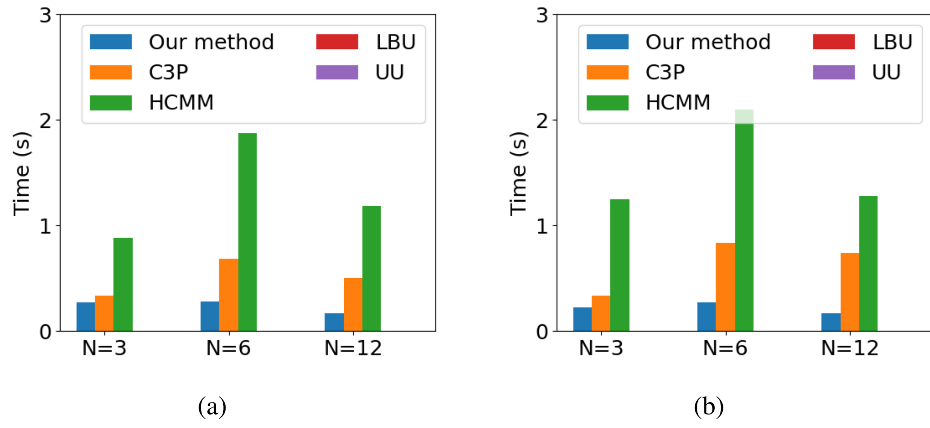


Figure 4.9. Average task completion times of different methods in different scenarios when there are a) one and b) two worker nodes leaving the NAC network.

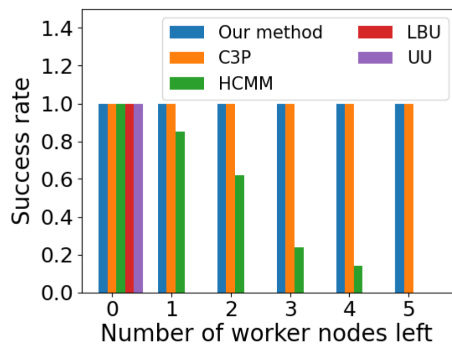


Figure 4.10. Success rates of different methods in Scenario 2 ($N = 6$) when an increasing number of worker nodes leave the NAC network.

4.7.4 Evaluation of Solution to $\mathscr{P}_2$

This section evaluates our solution to $\mathscr{P}_2$ for the NAC formation scenario with controllable UAVs.

Experiment Settings

We consider the following two computation scenarios.

- **Scenario 1:** $N = 3$, $p = 5000$.
- **Scenario 2:** $N = 6$, $p = 10000$.

The parameters of the reward function in (4.8) are set to $\omega = 15$ and $\omega_c = 5$. The target locations $\mathbf{g}_i, \forall i \in [N+1]$ are randomly sampled from $[-200m \times 200m] \times [-200m \times 200m]$. The bias term in the computation model is configured as $\xi = 0.04$.

We notice that our DRL method is limited in the scale of the NAC network it can handle, due to the exponentially growing state and action spaces. This is an issue inherent in all RL methods. One potential solution is to use MARL [178], but this requires the change of the computing architecture. We will leave this problem to the future work.

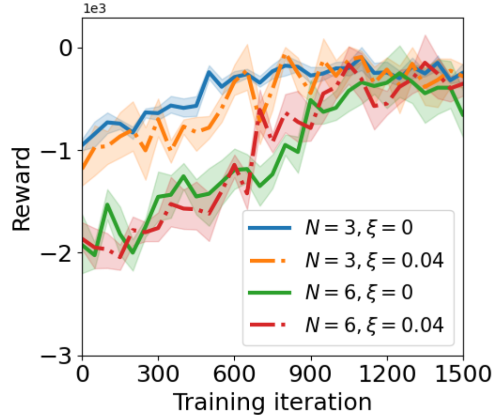


Figure 4.11. Training reward of our method for $\mathscr{P}_2$.

Training Reward

Fig. 4.11 shows the learning curves of our method under different settings. As we can see, the training rewards increase with more training iterations and finally converge.

Inference Time

The average inference time of our method measured over 50 runs is about 0.012s, which is small enough for UAVs to promptly react to potential collisions.

Comparative Results

Fig. 4.12(a) shows the total cost J_2 of different methods averaged over 100 experimental runs, where *our method (separate)* refers to the method that optimizes the two objectives of $\mathscr{P}_2$ separately, by using our solution to $\mathscr{P}_1$ for load allocation and the DRL algorithm described in Sec. 4.6.2 for UAV mobility control. As we can see, our method that jointly optimizes the two objectives outperforms all benchmark schemes in achieving the best tradeoff between computation efficiency and travel cost. It is noted that we can tune the weight ω to capture the relative importance of the two objectives depending on the application needs.

To better understand the performance of our method, we also plot in Fig. 4.12(b) and Fig. 4.12(c) the values of the total task completion time and total flight time, respectively. The results show that our method (joint) completes all computation tasks the most quickly, but

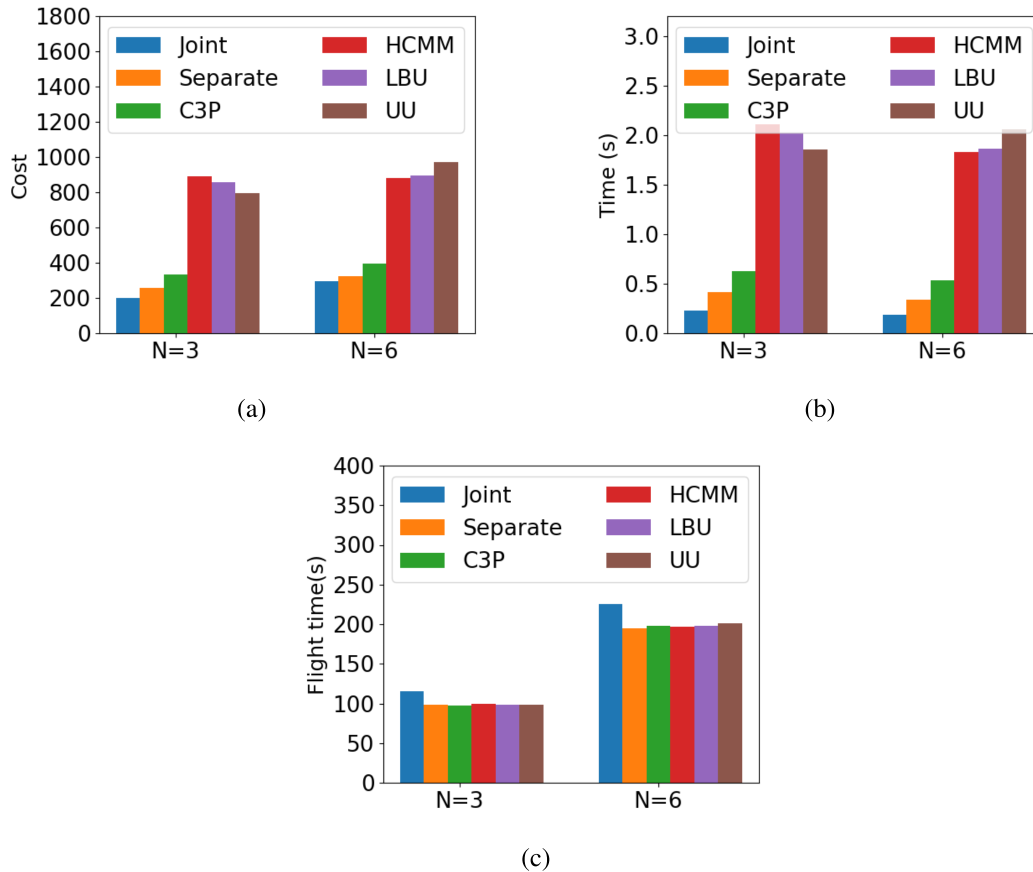


Figure 4.12. a) Total cost, b) average task completion times and c) total flight time of different methods in different scenarios.

consumes the highest UAV flight time. This is expected as all benchmark methods adopt the DRL policy that minimizes the flight time without considering the computing performance. Moreover, the comparison results between our method (joint) and our method (separate) confirm our hypothesis that the mobility of the UAVs can be proactively controlled to facilitate computing.

Fig. 4.13 plots the sample trajectories of the UAVs in Scenario 1 ($N = 3$) implementing different methods. The initial and target locations of the UAVs are marked using stars and diamonds, respectively. As we can see from Fig. 4.13(a), our method ensures that all UAVs will reach their target positions. Comparing Fig. 4.13(a) and Fig. 4.13(b), it is observed that the trajectories generated by benchmark methods are more straight than that generated by our method. This is because the benchmark methods optimize two objectives separately. Moreover, Fig. 4.14

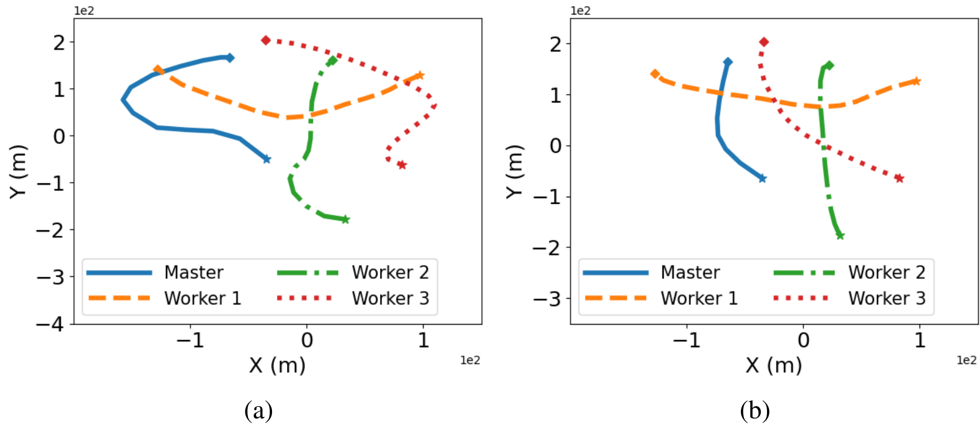


Figure 4.13. Sample trajectories of the UAVs in Scenario 1 by using a) our method with joint optimization; and b) benchmark methods.

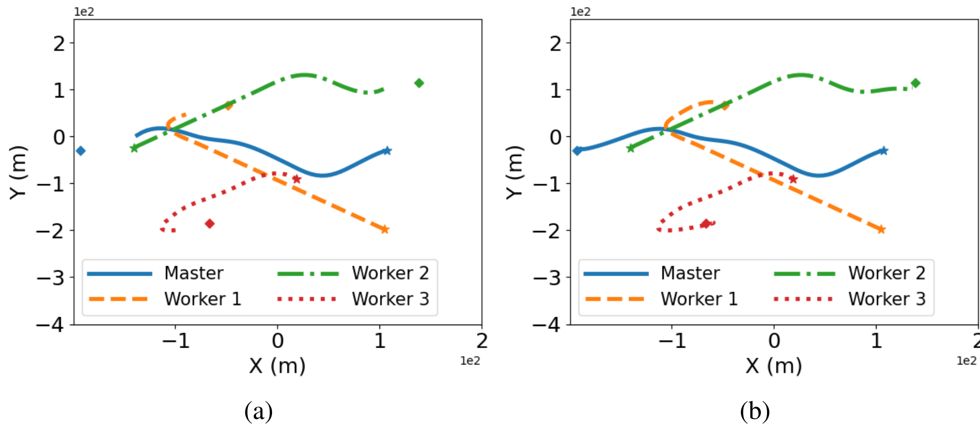


Figure 4.14. Sample trajectories of the UAVs in Scenario 1 a) when computation tasks are completed; and b) when the whole mission is completed.

illustrates how our method addresses the case when the computation tasks are completed before UAVs arrive at their target locations.

4.8 Conclusion

This chapter introduces innovative approaches to enable efficient, robust, and adaptable cooperative airborne computing in the dynamic, heterogeneous, and uncertain airspace. A CDC scheme, called D-BPCC, was first introduced that leverages the coding theory and a dynamic batch-processing based procedure to address the uncertainties in the dynamic and

heterogeneous NAC system. To optimize system performance, DRL based online decision-making strategies are then designed for two typical NAC formation scenarios, which do not rely on perfect communication, computation or UAV mobility models. Simulation results show that our methods are more resilient to uncertain system disturbances than existing solutions, including the UU, LBU, HCMM, and C3P schemes, and are adaptive to network topology and resource changes. Moreover, the effectiveness of our method in solving scenarios where NAC is formed by controllable UAVs demonstrates the benefits of UAV mobility control to robust computing. In the future, we will investigate MARL to address the scalability issue encountered by our DRL methods when the number of UAVs is large. We will also take energy consumption into the consideration.

4.9 Acknowledgement

This Chapter is a reprint of the accepted journal paper: B. Wang, J. Xie, K. Lu, Y. Wan, S. Fu, “Learning and Batch-Processing Based Coded Computation with Mobility Awareness for Networked Airborne Computing”, *IEEE Transactions on Vehicular Technology*, Nov. 2022.

Chapter 5

Coded Distributed Multi-Agent Reinforcement Learning with One-hop Neighbors

5.1 Introduction

In the previous two chapters (Chapters 3 and 4), we investigated how to achieve efficient and robust networked airborne computing in uncertain, heterogeneous, and dynamic airspace by exploring coded distributed computing techniques. In these studies, the simple and basic matrix multiplication problem was considered. In this chapter, we investigate a more complex computation problem, i.e., multi-agent reinforcement learning (MARL), and explore how to efficiently and robustly train MARL, even in large-scale settings, over the NAC systems.

Recent years have witnessed tremendous success of reinforcement learning (RL) in challenging decision making problems, such as robot control and video games. Research efforts are currently focused on multi-agent settings, including cooperative robot navigation [179], multi-player games [180], and traffic management [181]. Direct application of RL techniques in multi-agent settings by running single-agent algorithms simultaneously on each agent exhibits poor performance [178]. This is because, without considering interactions among the agents, the environment becomes non-stationary from the perspective of a single agent.

Multi-agent reinforcement learning (MARL) [182] addresses this challenge by considering all agents and their dynamics collectively when learning the value function and policy of an individual agent. Most effective MARL algorithms, such as multi-agent deep deterministic policy

gradient (MADDPG) [178] and counterfactual multi-agent (COMA) [183], adopt this strategy. However, learning a joint-state value or action-value (Q) or policy function is challenging due to the exponentially growing joint state and action spaces with increasing number of agents [184]. Policies trained with joint state-action pairs have poor performance in large-scale settings as demonstrated in recent work [185, 186] because their accurate approximation requires models with extremely large capacity.

MARL algorithms that improve the quality of learned policies for large-scale multi-agent settings often employ value function factorization, e.g., as in mean-field MARL (MFAC) [185] or scalable actor critic (SAC) [184], and training with an evolutionary population curriculum (EPC) [186]. While these methods achieve excellent performance, the training time can be significant when the number of agents increases because these methods cannot be easily trained in a distributed or parallel manner over multiple computers.

To address the challenge of training policies for large numbers of agents over a distributed computing architecture, we propose a MARL algorithm called Distributed multi-Agent Reinforcement Learning with One-hop Neighbors (DARL1N). DARL1N's *main advantage* over state-of-the-art MARL methods is that it allows distributed training, where each compute node simulates only a very small subset of the agent transitions. This is made possible by modeling the agent team topology as a proximity graph and representing the Q function and policy of each agent as a function of its one-hop neighbors only. This structure significantly reduces the representation complexity of the Q and policy functions and yet maintains expressiveness when training is done over varying states and numbers of neighbors. Furthermore, when agent interactions are restricted to one-hop neighborhoods, training an agent's Q function and policy requires transitions only of the agent itself and its potential two-hop neighbors. This enables highly efficient distributed training because each compute node needs to simulate only the transitions of the agents assigned to it and their two-hop neighbors.

RL or MARL policies can be trained over a distributed computing architecture either asynchronously or synchronously, which involves a central controller used to send and receive

value or policy parameters or gradients. Asynchronous training faces multiple challenges including slow convergence, difficult debugging and analysis, and sometimes subpar quality of learned policies as learners may return stale gradients evaluated with old parameters [187, 188, 189, 190, 191]. Synchronous training is superior in these aspects but is vulnerable to *straggler* compute nodes [156], caused by communication bottlenecks or software and hardware problems, that are prevalent in NAC system due to the high mobility of UAVs and airspace uncertainties, which lead to delays or failures in the training process. This chapter proposes a synchronous distributed learning architecture that mitigates the effects of stragglers in NAC system by employing *coding theory*.

Contributions: Our first contribution is a new MARL algorithm called DARL1N, which employs one-hop neighborhood factorization of the value and policy functions, allowing distributed training with each compute node simulating a small number of agent transitions. DARL1N supports *highly-efficient distributed training* and generates *high-quality multi-agent policies for large agent teams*. Our second contribution is a novel coded distributed learning architecture, which allows individual agents to be trained by multiple compute nodes simultaneously, enabling *resilience to stragglers*. Our analysis shows that introducing redundant computations via coding theory does not introduce bias in the value and policy gradient estimates. Our third contribution is a systematic investigation of *five state-of-the-art coding schemes for MARL training*, including MDS, Random Sparse, Repetition, LDPC, and LDGM codes. Moreover, we conduct comprehensive experiments comparing DARL1N with four state-of-the-art MARL methods, including MADDPG, MFAC, EPC and SAC, and evaluating their performance in different RL environments, including Ising Model, Food Collection, Grassland, Adversarial Battle, and Multi-Access Wireless Communication. We also conduct experiments to evaluate the resilience of DARL1N to stragglers when trained under different coding schemes.

5.2 Related Work

5.2.1 Multi-Agent Reinforcement Learning

Many MARL algorithms like MADDPG [178] adopt a centralized training and decentralized execution framework that considers other agents' behaviors when training an agent. However, learning accurate centralized value or Q functions under this framework becomes increasingly challenging when the number of agents increases due to exponentially growing joint and state and action spaces. This problem can be alleviated by factorizing the value or Q function into a combination of independent local value or Q functions that only depend on the local observations and actions of each agent. For example, in VDN [192], a full factorization of the Q function into a sum of local Q functions is employed. QMIX [193] improves VDN by combining the local Q functions monotonically using a mixing neural network, which makes it possible to represent more general Q functions without increasing the representation complexity significantly. QTRAN [194] further extends VDN and QMIX by factorizing a transformed value function without additivity and monotonicity assumptions.

The value or Q function can also be factorized according to a graph describing the agent team coordination. For instance, [195, 196] decomposed the Q function into a set of local Q functions with dependencies specified by agents that are connected in an undirected graph. [197] factorizes the Q function according to a coordination graph learned by a deep neural network. Mean-Field MARL [185] including Mean-Field Actor Critic (MFAC) and Mean-Field Q further approximates the factorized Q function by replacing the input actions with the mean value of neighboring agents' actions. Another method closely related to this chapter is SAC [184], which factorizes Q function of each agent using the states and actions of its κ-hop neighbors. The Q function factorization is similar to ours but SAC training cannot be distributed because it uses gradient descent with momentum which requires simultaneous simulation of multi-step transitions for many agents.

In addition to value factorization, there are several other methods proposed to enable

scalable MARL. MAAC [198] uses an attention module to abstract states of other agents when training an agent's Q function, which reduces the quadratically increasing input space to a linear space. EPC [186] applies curriculum learning to gradually scale MARL up. It adopts population invariant policy and Q functions represented with attention modules to support varying numbers of agents in different learning stages. To further speed up training, EPC is implemented in a parallel computing architecture that trains agents and simulates environments in parallel processes.

5.2.2 Distributed and Parallel Architectures for RL and MARL

Multiple distributed or parallel architectures have been proposed to accelerate RL and MARL training. The first massively parallel architecture was presented in [199] to train deep Q networks. It creates multiple actors and learners, with each actor interacting with its environment independently and each learner updating the parameters. This architecture, however, only works for off-policy RL algorithms. To support both off-policy and on-policy RL algorithms, a more general parallel architecture called A3C [200] was then developed. In A3C, multiple compute nodes run in parallel to train agent's policy asynchronously. Extensions of A3C include A2C [190], which is a synchronous version of A3C, and GA3C [201], which is a GPU version of A3C. A3C has also been extended to multi-agent settings [202], where each computing instance performs independent training for all agent parameters asynchronously. Also of interest is the distributed architecture presented in [203] which was designed to reduce the communication overhead between learners and the central controller by only communicating significant gradients. The main focus of this chapter is to handle the presence of stragglers in distributed computing systems, which has not been considered by existing distributed and parallel methods for training RL and MARL algorithms.

5.2.3 Coded Distributed Computing

Coded computation has recently gained increasing popularity as a promising approach to mitigate straggler effects in distributed computing systems [115]. It was first proposed in [115] to speed up the computation of distributed matrix multiplication in the presence of stragglers, and has since been extended to accelerate other computation tasks such as linear inverse problems [158], convolution [204], and map reduce [161]. Applying coded computation to the training of distributed MARL algorithms has not been investigated thoroughly. In our previous work [162], we conducted a preliminary study on the merits of coded computation in speeding up the training of MADDPG [178] in a distributed manner. In this work, we extend our formulation and apply it to the DARL1N algorithm, which is designed to support distributed computation by limiting the number of operations that need to be performed at each compute node.

5.3 Background

This section introduces the MARL problem. In MARL, M agents learn to optimize their behavior by interacting with the environment. Denote the state and action of agent $i \in [M] := \{1, \ldots, M\}$ by $s_i \in \mathscr{S}_i$ and $a_i \in \mathscr{A}_i$, respectively, where $\mathscr{S}_i$ and $\mathscr{A}_i$ are the corresponding state and action spaces. Let $\mathbf{s} := (s_1, \ldots, s_M) \in \mathscr{S} := \prod_{i \in [M]} \mathscr{S}_i$ and $\mathbf{a} := (a_1, \ldots, a_M) \in \mathscr{A} := \prod_{i \in [M]} \mathscr{A}_i$ denote the joint state and action of all agents. At time t, a joint action $\mathbf{a}(t)$ applied at state $\mathbf{s}(t)$ triggers a transition to a new state $\mathbf{s}(t+1) \in \mathscr{S}$ according to a conditional probability density function (pdf) $p(\mathbf{s}(t+1)|\mathbf{s}(t), \mathbf{a}(t))$. After each transition, each agent i receives a reward $r_i(\mathbf{s}(t), \mathbf{a}(t))$, determined by the joint state and action according to the function $r_i : \mathscr{S} \times \mathscr{A} \mapsto \mathbb{R}$.

The objective of each agent i is to design a policy $\mu_i : \mathscr{S} \to \mathscr{A}_i$ to maximize the expected cumulative discounted reward:

$$V_i^{\mu}(\mathbf{s}) := \mathbb{E}_{\substack{\mathbf{a}(t)=\mu(\mathbf{s}(t)) \\ \mathbf{s}(t) \sim p}} \left[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}(t), \mathbf{a}(t)) \mid \mathbf{s}(0) = \mathbf{s} \right],$$

where $\mu := (\mu_1, \ldots, \mu_M)$ denotes the joint policy of all agents and $\gamma \in (0,1)$ is a discount factor. The function $V_i^{\mu}(\mathbf{s})$ is known as the *value function* of agent i associated with joint policy μ. Many RL and MARL techniques consider stochastic policies to support the exploration-exploitation trade-off when approximating the value and policy functions [205]. However, since optimal policies are known to be deterministic, it is possible to directly restrict attention to deterministic policies during the learning process, e.g., as done in DDPG [206] and MADDPG [178].

An optimal policy μ_i^* for agent i can also be obtained by maximizing the *action-value (Q) function*:

$$Q_i^{\mu}(\mathbf{s},\mathbf{a}) := \mathbb{E}_{\substack{\mathbf{a}(t)=\mu(\mathbf{s}(t)) \\ \mathbf{s}(t)\sim p}}\left[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}(t),\mathbf{a}(t)) \mid \mathbf{s}(0)=\mathbf{s}, \mathbf{a}(0)=\mathbf{a}\right]$$

and setting $\mu_i^*(\mathbf{s}) \in \arg\max_{a_i} \max_{\mathbf{a}_{-i}} Q_i^*(\mathbf{s},\mathbf{a})$, where $Q_i^*(\mathbf{s},\mathbf{a}) := \max_{\mu} Q_i^{\mu}(\mathbf{s},\mathbf{a})$ and $\mathbf{a}_{-i}$ denotes the actions of all agents except i. In the rest of the chapter, we omit the time notation t for simplicity, when there is no risk of confusion.

5.4 Problem Statement

To develop a distributed MARL algorithm, we impose additional structure on the MARL problem. Assume that all agents share a common state space, i.e., $\mathscr{S}_i = \mathscr{S}_j$, $\forall i, j \in [M]$ and let $\text{dist} : \mathscr{S}_i \times \mathscr{S}_i \to \mathbb{R}$ be a distance metric on the state space.

Consider a proximity graph [207] that models the topology of the agent team. A d-disk proximity graph is defined as a mapping that associates the joint state $\mathbf{s} \in \mathscr{S}$ with an undirected graph $(\mathscr{V}, \mathscr{E})$ such that $\mathscr{V} = \{s_1, s_2, \ldots, s_M\}$ and $\mathscr{E} = \{(s_i, s_j) | \text{dist}(s_i, s_j) \leq d, i \neq j\}$. Define the set of *one-hop neighbors* of agent i as $\mathscr{N}_i := \{j | (s_i, s_j) \in \mathscr{E}\} \cup \{i\}$. We make the following assumption about the agents' motion.

Assumption 1. *The distance between two consecutive states, $s_i(t)$ and $s_i(t+1)$, of agent i is bounded, i.e., $\text{dist}(s_i(t), s_i(t+1)) \leq \varepsilon$, for some $\varepsilon > 0$.*

This assumption is satisfied in many problems where, e.g., due to physical constraints, the agent states can only change by a bounded amount in a single time step.

Define the set of *potential neighbors* of agent i at time t as $\mathscr{P}_i(t) := \{j | \operatorname{dist}(s_j(t), s_i(t)) \leq 2\varepsilon + d\}$, which captures the set of agents that may become one-hop neighbors of agent i at time $t+1$. Denote the joint state and action of the one-hop neighbors of agent i by $\mathbf{s}_{\mathscr{N}_i} = (s_{j_1}, \ldots, s_{j_{|\mathscr{N}_i|}})$ and $\mathbf{a}_{\mathscr{N}_i} = (a_{j_1}, \ldots, a_{j_{|\mathscr{N}_i|}})$, respectively, where $j_1, \ldots, j_{|\mathscr{N}_i|} \in \mathscr{N}_i$. Our key idea is to let agent i's policy, $a_i = \mu_i(\mathbf{s}_{\mathscr{N}_i})$, only depend on the one-hop neighbor states $\mathbf{s}_{\mathscr{N}_i}$ instead of all agent states $\mathbf{s}$. The intuition is that agents that are far away from agent i at time t have little impact on its current action $a_i(t)$. To emphasize that the output of a function $f : \prod_{i \in [M]} \mathscr{S}_i \mapsto \mathbb{R}$ is affected only by a subset $\mathscr{N} \subseteq [M]$ of the input dimensions, we use the notation $f(\mathbf{s}) = f(\mathbf{s}_{\mathscr{N}})$ for $\mathbf{s} \in \mathscr{S}$ and $\mathbf{s}_{\mathscr{N}} \in \prod_{i \in \mathscr{N}} \mathscr{S}_i$ in the remainder of the chapter. We make two additional assumptions on the problem structure to ensure the validity of our policy model.

Assumption 2. *The reward of agent i can be fully specified using its one-hop neighbor states $\mathbf{s}_{\mathscr{N}_i}$ and actions $\mathbf{a}_{\mathscr{N}_i}$, i.e., $r_i(\mathbf{s}, \mathbf{a}) = r_i(\mathbf{s}_{\mathscr{N}_i}, \mathbf{a}_{\mathscr{N}_i})$ and its absolute value is upper bounded by $|r_i(\mathbf{s}_{\mathscr{N}_i}, \mathbf{a}_{\mathscr{N}_i})| \leq \bar{r}$, for some $\bar{r} > 0$.*

Assumption 2 is satisfied in many multi-agent problems where the reward of one agent is determined only by the states and actions of nearby agents. Examples are provided in Sec. 5.8. Similar assumptions are adopted in [184, 208, 209].

Assumption 3. *The transition model of agent i depends only on its action a_i and one-hop neighbor states $\mathbf{s}_{\mathscr{N}_i}$, i.e., $p_i\left(s_i(t+1) \mid \mathbf{s}(t), a_i(t)\right) = p_i\left(s_i(t+1) \mid \mathbf{s}_{\mathscr{N}_i}(t), a_i(t)\right)$.*

Assumption 3 is common for multi-agent networked systems as in [184, 209]. As a result, the joint state transition pdf decomposes as:

$$p(\mathbf{s}(t+1) \mid \mathbf{s}(t), \mathbf{a}(t)) = \prod_{i=1}^{M} p_i\left(s_i(t+1) \mid \mathbf{s}_{\mathscr{N}_i}(t), a_i(t)\right).$$

The objective of each agent i is to obtain an optimal policy μ_i^* by solving the following problem:

$$\mu_i^*(\mathbf{s}_{\mathcal{N}_i}) = \arg\max_{a_i} \max_{\mathbf{a}_{-i}} Q_i^*(\mathbf{s}, \mathbf{a}), \tag{5.1}$$

where $Q_i^*(\mathbf{s}, \mathbf{a}) := \max_\mu Q_i^\mu(\mathbf{s}, \mathbf{a})$ is the optimal action-value (Q) function introduced in the previous section.

The goal of this chapter is to develop a MARL algorithm that (i) utilizes policy and value representations that scale favorably with the number of agents M and (ii) allows efficient training on a distributed computing system containing straggler compute nodes, which are slow or unresponsive.

To analyze the performance of distributed training algorithms in the presence of stragglers, we make the following general assumption.

Assumption 4. *In each training iteration, each compute node in a distributed computing system has a probability of $\eta \in [0, 1]$ to become a straggler that slows down computation or fails completely.*

5.5 Distributed multi-Agent Reinforcement Learning with One-hop Neighbors

This section develops the DARL1N algorithm to solve the MARL problem with proximity graph structure introduced in Sec. 5.4. Instead of considering global agent interactions, DARL1N only considers the effect of the one-hop neighbors of an agent in representing its Q and policy functions. This allows updating the Q and policy function parameters using only local one-hop neighborhood transitions.

Specifically, the Q function of each agent i can be expressed as a function of its one-hop neighbor states $\mathbf{s}_{\mathcal{N}_i}$ and actions $\mathbf{a}_{\mathcal{N}_i}$ as well as the states $\mathbf{s}_{\mathcal{N}_i^-}$ and actions $\mathbf{a}_{\mathcal{N}_i^-}$ of the remaining

agents that are not immediate neighbors of i:

$$Q_i^{\mu}(\mathbf{s},\mathbf{a}) = Q_i^{\mu}(\mathbf{s}_{\mathcal{N}_i},\mathbf{s}_{\mathcal{N}_i^-},\mathbf{a}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i^-}). \tag{5.2}$$

Inspired by the SAC algorithm [184], we approximate the Q value with a function $\tilde{Q}_i^{\mu}$ that depends only on one-hop neighbor states and actions:

$$\tilde{Q}_i^{\mu}(\mathbf{s}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i}) = \sum_{\mathbf{s}_{\mathcal{N}_i^-},\mathbf{a}_{\mathcal{N}_i^-}} w_i(\mathbf{s}_{\mathcal{N}_i},\mathbf{s}_{\mathcal{N}_i^-},\mathbf{a}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i^-}) Q_i^{\mu}(\mathbf{s}_{\mathcal{N}_i},\mathbf{s}_{\mathcal{N}_i^-},\mathbf{a}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i^-})$$

where the weights $w_i(\mathbf{s}_{\mathcal{N}_i},\mathbf{s}_{\mathcal{N}_i^-},\mathbf{a}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i^-}) > 0$ satisfy $\sum_{\mathbf{s}_{\mathcal{N}_i^-},\mathbf{a}_{\mathcal{N}_i^-}} w_i(\mathbf{s}_{\mathcal{N}_i},\mathbf{s}_{\mathcal{N}_i^-},\mathbf{a}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i^-}) = 1$. The approximation error is given in the following lemma.

Lemma 9. *Under Assumptions 2 and 3, the approximation error between $\tilde{Q}_i^{\mu}(\mathbf{s}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i})$ and $Q_i^{\mu}(\mathbf{s},\mathbf{a})$ is bounded by:*

$$|\tilde{Q}_i^{\mu}(\mathbf{s}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i}) - Q_i^{\mu}(\mathbf{s},\mathbf{a})| \leq \frac{2\bar{r}\gamma}{1-\gamma}.$$

Proof. See Appendix B.1. □

We parameterize the approximated Q function $\tilde{Q}_i^{\mu}(\mathbf{s}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i})$ and the policy $\mu_i(\mathbf{s}_{\mathcal{N}_i})$ by θ_i and ϕ_i, respectively. To handle the varying sizes of $\mathbf{s}_{\mathcal{N}_i}$ and $\mathbf{a}_{\mathcal{N}_i}$, in the implementation, we let the input dimension of $\tilde{Q}_i^{\mu}$ to be the largest possible dimension of $(\mathbf{s}_{\mathcal{N}_i},\mathbf{a}_{\mathcal{N}_i})$, and apply zero-padding for agents that are not within the one-hop neighborhood of agent i. The same procedure is applied to represent $\mu_i(\mathbf{s}_{\mathcal{N}_i})$. Implementation details are provided in Sec. 5.8.

To learn the approximated Q function $\tilde{Q}_i^{\mu}$, instead of incremental on-policy updates to the Q function as in SAC [184], we apply off-policy temporal-difference learning with a buffer similar to MADDPG [178]. The parameters θ_i of the approximated Q function are updated by

minimizing the following temporal difference error:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_l'}\}_{\forall l \in \mathcal{N}_i'}) \sim \mathcal{D}_i} \left[\left(\tilde{Q}_i^{\mu} \left(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i} \right) - y \right)^2 \right]$$
$$y = r_i + \gamma \hat{Q}_i^{\hat{\mu}} \left(\mathbf{s}_{\mathcal{N}_i'}, \mathbf{a}_{\mathcal{N}_i'} \right) \quad (5.3)$$

where $\mathcal{D}_i$ is a replay buffer for agent i that contains information only from $\mathcal{N}_i$ and $\mathcal{N}_i'$, the one-hop neighbors of agent i at the current and next time steps, and the one-hop neighbors $\mathcal{N}_l'$ for $l \in \mathcal{N}_i'$. To stabilize the training, a target Q function $\hat{Q}_i^{\hat{\mu}}$ with parameters $\hat{\theta}_i$ and a target policy function $\hat{\mu}_i$ with parameters $\hat{\phi}_i$ are used. The parameters $\hat{\theta}_i$ and $\hat{\phi}_i$ are updated using Polyak averaging:

$$\hat{\theta}_i = \tau \hat{\theta}_i + (1 - \tau)\theta_i,$$
$$\hat{\phi}_i = \tau \hat{\phi}_i + (1 - \tau)\phi_i,$$

where τ is a hyperparameter. In contrast to MADDPG [178], the replay buffer $\mathcal{D}_i$ for agent i only needs to store its local interactions $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_l'}\}_{\forall l \in \mathcal{N}_i'})$ with nearby agents. Note that $\{\mathbf{s}_{\mathcal{N}_l'}\}_{\forall l \in \mathcal{N}_i'}$ is used to calculate $\mathbf{a}_{\mathcal{N}_i'}$. Also, in contrast to SAC [184], each agent i only needs to collect its own training data by simulating local two-hop interactions. This allows efficient distributed training as we explain in Sec. 5.6. Agent i's policy parameters ϕ_i are updated using a gradient

$$\mathbf{g}(\phi_i) = \mathbb{E}_{\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i} \sim \mathcal{D}_i} \left[\nabla_{\phi_i} \mu_i \left(\mathbf{s}_{\mathcal{N}_i} \right) \nabla_{a_i} \tilde{Q}_i^{\mu} \left(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i} \right) \right], \quad (5.4)$$

where again data $\mathcal{D}_i$ only from local interactions is needed.

To implement the parameter updates proposed above, agent i needs training data $\mathcal{D}_i = (\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_l'}\}_{l \in \mathcal{N}_i'})$ from its one-hop neighbors at the current and next time steps. The relation between one-hop neighbors at the current and next time steps is captured by the following proposition.

Proposition 1. *Under Assumption 1, if an agent j is not a potential neighbor of agent i at time t,*

i.e., $j \notin \mathscr{P}_i(t)$*, it will not be a one-hop neighbor of agent i at time* $t+1$*, i.e.,* $j \notin \mathscr{N}_i(t+1)$.

Proof. See Appendix B.2. □

Proposition 1 allows us to decouple the global interactions among agents and limit the necessary observations to be among one-hop neighbors. To collect training data, at each time step, agent i first interacts with its one-hop neighbors to obtain their states $\mathbf{s}_{\mathscr{N}_i}$ and actions $\mathbf{a}_{\mathscr{N}_i}$, and compute its reward $r_i(\mathbf{s}_{\mathscr{N}_i}, \mathbf{a}_{\mathscr{N}_i})$. To obtain $\mathbf{s}_{\mathscr{N}_l'}$ for all $l \in \mathscr{N}_i'$, we first determine agent i's one-hop neighbors at the next time step, $\mathscr{N}_i'$. Using Proposition 1, we let each potential neighbor $k \in \mathscr{P}_i$ perform a transition to a new state $s_k' \sim p_k(\cdot|\mathbf{s}_{\mathscr{N}_k}, a_k)$, which is sufficient to determine $\mathscr{N}_i'$. Then, we let the potential neighbors $\mathscr{P}_l$ of each new neighbor $l \in \mathscr{N}_i'$ perform transitions to determine $\mathscr{N}_l'$ and obtain $\mathbf{s}_{\mathscr{N}_l'}$.

Fig. 5.1(a) illustrates the data collection process. At time t, agent i obtains $\mathbf{s}_{\mathscr{N}_i}$, $\mathbf{a}_{\mathscr{N}_i}$, and $r_i(\mathbf{s}_{\mathscr{N}_i}, \mathbf{a}_{\mathscr{N}_i})$ for $\mathscr{N}_i = \{i, 1\}$. Then, the potential neighbors of agent i, $\mathscr{P}_i = \{1, 2, i\}$, proceed to their next states at time $t+1$. This is sufficient to determine that $\mathscr{N}_i' = \{i, 2\}$ and obtain $\mathbf{s}_{\mathscr{N}_i'}$. Finally, we let agent 3, which belongs to set $\mathscr{P}_2 = \{i, 1, 2, 3\}$, perform a transition to determine that $\mathscr{N}_2' = \{i, 2, 3\}$ and obtain $\mathbf{s}_{\mathscr{N}_2'}$.

As each agent only needs to interact with one-hop neighbors to update its parameters, the agents can be trained in parallel on a distributed computing architecture, where each compute node only needs to simulate the two-hop neighbor transitions for agents assigned to it for training.

5.6 Coded Distributed Learning Architecture

In this section, we introduce an efficient and resilient distributed computing architecture for training DARL1N, which significantly accelerates the training speed, especially for large agent teams, and mitigates the effect of stragglers in distributed computing systems by leveraging coding theory.

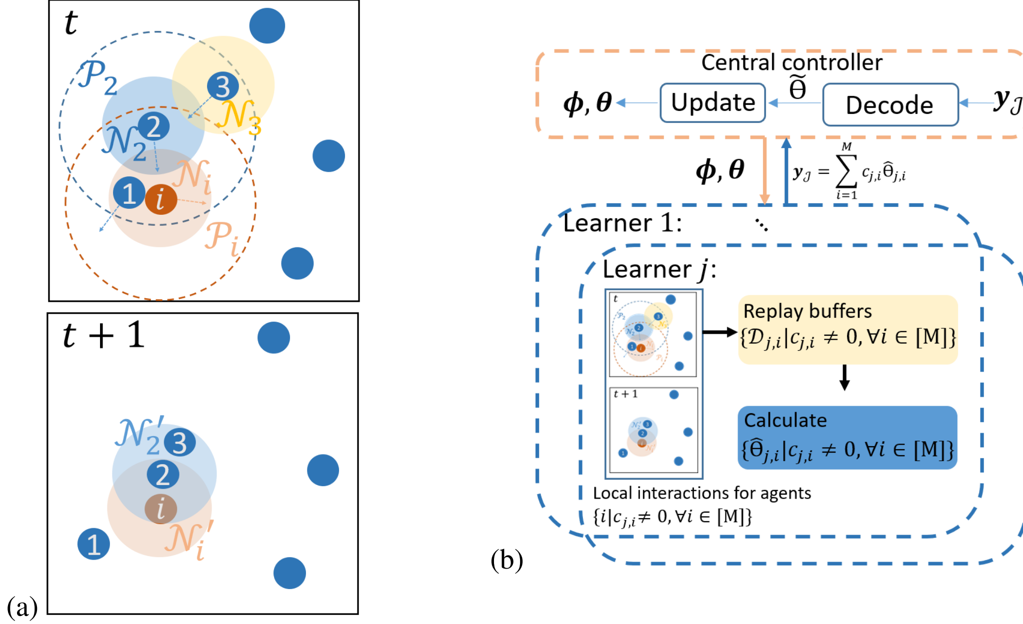


Figure 5.1. (a) One-hop neighbor transitions from one time step to the next in a d-disk proximity graph; (b) Coded distributed learning architecture.

5.6.1 Coded Distributed Learning Architecture

A coded distributed learning architecture, illustrated in Fig. 5.1(b), consists of a central controller and N computation nodes, called *learners*. The central controller stores a copy of all parameters of the policy ϕ_i, target policy $\hat{\phi}_i$, Q function θ_i, and target Q function $\hat{\theta}_i$, for all $i \in [M]$. In each training iteration, the central controller broadcasts all agents' parameters to all learners, who then calculate and return the gradients required for updating the parameters. In a traditional uncoded distributed architecture, each agent is only trained (with its policy and value gradients computed) by a single learner node. If any learner becomes slow or unresponsive, the whole training procedure is delayed or may fail. Our coded distributed learning architecture addresses the possible presence of stragglers in the computing system by introducing redundant computations. We let more than one learner train each agent, which not only improves the system resilience to stragglers and but also accelerates the training speed, as we show in Sec. 5.8.2. To describe which learners are assigned to train each agent, we introduce an assignment matrix $\mathbf{C} \in \mathbb{R}^{N \times M}$ with non-zero entries $c_{j,i} \neq 0$ indicating that learner $j \in [N]$ is assigned to train

agent $i \in [M]$. The complete set of learners assigned to train an agent i can then be determined by $\{j | c_{j,i} \neq 0, \forall j \in [N]\}$. To construct the assignment matrix $\mathbf{C}$, we apply coding theory as explained in Sec.5.7.

To calculate the gradients for an agent i, each learner j with $c_{j,i} \neq 0$ simulates transitions to get the interaction data $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_l'}\}_{l \in \mathcal{N}_i'})$ as described in Sec. 5.5, which are stored in a replay buffer $\mathscr{D}_{j,i}$. After that, learner j calculates the gradients of the temporal difference error needed for updating the Q function parameters θ_i of agent i using (5.3) and updating the policy parameters ϕ_i using (5.4).

As the replay buffer $\mathscr{D}_{j,i}$ can have a large size, to improve efficiency, we use a mini-batch $\mathscr{B}_{j,i}$ uniformly sampled from $\mathscr{D}_{j,i}$ to estimate the expectations in (5.3)-(5.4). In particular, the temporal difference error in (5.3) is estimated with:

$$\hat{\mathscr{L}}_j(\theta_i) = \frac{1}{|\mathscr{B}_{j,i}|} \sum_{\substack{(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, r_i, \{\mathbf{s}_{\mathcal{N}_l'}\}_{\forall l \in \mathcal{N}_i'}) \\ \in \mathscr{B}_{j,i}}} \left[\left(\tilde{Q}_i^{\mu} \left(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i} \right) - y \right)^2 \right]$$

$$y = r_i + \gamma \hat{Q}_i^{\hat{\mu}} \left(\mathbf{s}_{\mathcal{N}_i'}, \mathbf{a}_{\mathcal{N}_i'} \right). \tag{5.5}$$

Similarly, the gradients used to update policy parameters are estimated with:

$$\hat{\mathbf{g}}_j(\phi_i) = \frac{1}{|\mathscr{B}_{j,i}|} \sum_{\substack{(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}) \\ \in \mathscr{B}_{j,i}}} \left[\nabla_{\phi_i} \mu_i \left(\mathbf{s}_{\mathcal{N}_i} \right) \nabla_{a_i} \tilde{Q}_i^{\mu} \left(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i} \right) \right]. \tag{5.6}$$

Let $\hat{\mathbf{e}}_{j,i} = [\nabla \hat{\mathscr{L}}_j(\theta_i), \hat{\mathbf{g}}_j(\phi_i)]$ denote the concatenation of estimated gradients. Instead of directly returning the estimated gradients for all agents trained by learner j, i.e., $\{\hat{\mathbf{e}}_{j,i} | \forall i \in [M], c_{j,i} \neq 0\}$, learner j calculates a linear combination of the gradients:

$$y_j = \sum_{i=1}^{M} c_{j,i} \hat{\mathbf{e}}_{j,i}$$

with weights provided by the assignment matrix $\mathbf{C}$ and returns y_j back to the central controller.

At the central controller, let $\mathbf{y}_{\mathscr{J}}$ denote the results that have arrived by a certain time from learners $\mathscr{J} = \{j|y_j \text{ is received}\}$. Moreover, let $\mathbf{C}_{\mathscr{J}} \in \mathbb{R}^{|\mathscr{J}|\times M}$ be a submatrix of $\mathbf{C}$ formed by the j-th row of $\mathbf{C}, \forall j \in \mathscr{J}$. The received gradients $\mathbf{y}_{\mathscr{J}}$ satisfy:

$$
\begin{aligned}
\mathbf{y}_{\mathscr{J}} &= \mathbf{D}\mathbf{q} \\
\mathbf{D} &= \text{diag}(C_{\mathscr{J},1}, C_{\mathscr{J},2}, \ldots, C_{\mathscr{J},|\mathscr{J}|}) \\
\mathbf{q} &= [\hat{\mathbf{e}}_{1,1}, \hat{\mathbf{e}}_{1,2}, \ldots, \hat{\mathbf{e}}_{N,M-1}, \hat{\mathbf{e}}_{N,M}],
\end{aligned} \tag{5.7}
$$

where diag() creates a block-diagonal matrix with the i-th rows of $C_{\mathscr{J}}$, denoted as $C_{\mathscr{J},i}$, on its diagonal. The vector $\mathbf{q}$ is a concatenation of all the gradients estimated by all the learners. The central controller updates the agents' parameters once it receives enough results to decode all estimated gradients, denoted as $\tilde{\mathbf{e}}$. This happens when $\text{rank}(\mathbf{C}_{\mathscr{J}}) = M$, and the decoding equation is given as follows:

$$
\tilde{\mathbf{e}} = (\mathbf{C}_{\mathscr{J}}^T \mathbf{C}_{\mathscr{J}})^{-1} \mathbf{C}_{\mathscr{J}}^T \mathbf{y}_{\mathscr{J}}. \tag{5.8}
$$

Alg. 3 summarizes the coded training procedure of DARL1N over a distributed computing architecture.

5.6.2 Assessment of Gradient Estimator

In our coded distributed learning architecture, the gradients $\tilde{\mathbf{e}}$ used by the central controller for parameter updates are estimates of the true gradients $\mathbf{e} = [\mathbf{e}_1, \ldots, \mathbf{e}_M]$, where $\mathbf{e}_i = [\nabla \mathscr{L}(\theta_i), \mathbf{g}(\phi_i)]$ with $\mathscr{L}(\theta_i)$ and $\mathbf{g}(\phi_i)$ defined in (5.4) and (5.3), respectively. In this section, we evaluate the bias and variance of this gradient estimator.

We calculate the bias of the gradient estimator $\tilde{\mathbf{e}}$ using (5.7) and (5.8):

$$
\begin{aligned}
\mathbb{E}[\tilde{\mathbf{e}}] - \mathbf{e} &= (\mathbf{C}_{\mathscr{J}}^T \mathbf{C}_{\mathscr{J}})^{-1} \mathbf{C}_{\mathscr{J}}^T \mathbb{E}[\mathbf{y}_{\mathscr{J}}] - \mathbf{e} \\
&= (\mathbf{C}_{\mathscr{J}}^T \mathbf{C}_{\mathscr{J}})^{-1} \mathbf{C}_{\mathscr{J}}^T \mathbf{D} \mathbb{E}[\mathbf{q}] - \mathbf{e}.
\end{aligned} \tag{5.9}
$$

Since each learner uses the same set of parameters broadcast by the central controller for agent-environment interaction in each training iteration, the replay buffers $\mathscr{D}_{j,i}, \forall j \in [N]$, all follow the same distribution as that of $\mathscr{D}_i$. Therefore, we have $\mathbb{E}[\hat{\mathbf{e}}_{j,i}] = \mathbf{e}_i$ and $\mathbf{D}\mathbb{E}[\mathbf{q}] = \mathbf{C}_{\mathscr{I}}\mathbf{e}$ leading to:

$$\mathbb{E}[\tilde{\mathbf{e}}] - \mathbf{e} = (\mathbf{C}^T_{\mathscr{I}}\mathbf{C}_{\mathscr{I}})^{-1}\mathbf{C}^T_{\mathscr{I}}\mathbf{C}_{\mathscr{I}}\mathbf{e} - \mathbf{e} = 0, \tag{5.10}$$

which shows that $\tilde{\mathbf{e}}$ is an unbiased estimator.

Next, we compute the variance of the gradient estimator $\tilde{\mathbf{e}}$:

$$\begin{aligned} \text{Var}[\tilde{\mathbf{e}}] &= \text{Var}[(\mathbf{C}^T_{\mathscr{I}}\mathbf{C}_{\mathscr{I}})^{-1}\mathbf{C}^T_{\mathscr{I}}\mathbf{y}_{\mathscr{I}}] \\ &= (\mathbf{C}^T_{\mathscr{I}}\mathbf{C}_{\mathscr{I}})^{-1}\mathbf{C}^T_{\mathscr{I}}\mathbf{D}\text{Var}(\mathbf{q})\mathbf{D}^T((\mathbf{C}^T_{\mathscr{I}}\mathbf{C}_{\mathscr{I}})^{-1}\mathbf{C}^T_{\mathscr{I}})^T, \end{aligned} \tag{5.11}$$

where $\text{Var}[\mathbf{q}] = \text{diag}(\text{Var}(\hat{\mathbf{e}}_{1,1}), \ldots, \text{Var}(\hat{\mathbf{e}}_{N,M}))$, since $\hat{\mathbf{e}}_{i,j}, \forall i \in [M]$ are independent from each other for each $j \in [N]$. According to (5.11), we can see that the variance of the gradient estimator $\tilde{\mathbf{e}}$ is impacted by $\mathbf{C}_{\mathscr{I}}$, which is determined by the assignment matrix $\mathbf{C}$ as well as the learners who return their computations promptly. The impact of the gradient estimator variance on the training performance will be evaluated empirically in Sec. 5.8.2. In the following section, we investigate the construction of assignment matrix $\mathbf{C}$.

5.7 Assignment Matrix Construction

In this section, we introduce different schemes, both uncoded and coded, to construct the assignment matrix $\mathbf{C}$. Five schemes that have been used for enhancing the resilience of distributed computing systems are investigated.

5.7.1 Uncoded Assignment Scheme

In an uncoded distributed training architecture, different learner nodes train different agents exclusively. The assignment matrix for the uncoded scheme assigns a single agent to each

Algorithm 3: Coded DARL1N

```
   // Central controller:
1  Initialize policy, target policy, Q, and target Q parameters φ = {φ_i, φ̂_i}_{i∈[M]}, θ = {θ_i, θ̂_i}_{i∈[M]};
2  Broadcast φ, θ to the learners;
3  y_𝒥 ← [ ];
4  do
5  |  Listen to channel and collect y_j from the learners: y_𝒥 ← [y_𝒥, y_j], j ∈ [N];
6  while ẽ is not recoverable;
7  Send acknowledgements to learners;
8  Update φ, θ with ẽ;
   // Learner j:
9  Initialize replay buffer 𝒟_{j,i};
10 for iter = 1 : max_iteration do
11 |  Listen to channel;
12 |  if φ, θ received from the central controller then
13 |  |  y_j ← 0; i ← 1;
14 |  |  while i ≤ M and no acknowledgement received do
15 |  |  |  if c_{j,i} ≠ 0 then
   |  |  |  |  // Local interactions:
16 |  |  |  |  Perform local interactions to collect training data for agent i and store the
   |  |  |  |    data into 𝒟_{j,i};
17 |  |  |  |  Sample a mini-batch from 𝒟_{j,i} and calculate ê_{j,i} using (5.5)-(5.6);
18 |  |  |  y_j ← y_j + c_{j,i} ê_{j,i};
19 |  |  |  i ← i + 1;
20 |  |  |  Send updated y_j to the central controller;
```

of the first M learners and no agents to the remaining learners:

$$C^{\text{Uncoded}} = [\mathbf{I}_M|\mathbf{0}]^T. \tag{5.12}$$

The next proposition shows the probability that the uncoded scheme will be influenced, i.e., delayed or failed, by randomly occurring straggler nodes.

Proposition 2. *Under Assumption 4 with* $\mathbf{C}$ *in* (5.12)*, the probability that the distributed training procedure in Alg. 3 will be influenced by stragglers is* $1-(1-\eta)^M$.

Remark 3. *Under the Uncoded scheme* (5.12)*, the distributed training procedure fails whenever one or more learners fail to return their gradient computations at any training iteration since the*

central controller requires results from all learners to update the agents' parameters.

5.7.2 Coded Assignment Schemes

Coded distributed training assigns each agent to multiple learners. Different codes lead to different assignment matrices $\mathbf{C}$ with different properties. We introduce five assignment schemes designed based on five commonly used codes.

MDS Code

A Maximum Distance Separable code [210] is an erasure code with the property that any square submatrix of its assignment matrix $\mathbf{C}^{\mathrm{MDS}}$ has full rank. There are multiple ways to construct MDS assignment matrices. One common way is to use a Vandermonde matrix [211]:

$$\mathbf{C}^{\mathrm{MDS}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_M \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_M^2 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{N-1} & \alpha_2^{N-1} & \cdots & \alpha_M^{N-1} \end{bmatrix}, \tag{5.13}$$

where $\alpha_i \in \mathbb{R}$, $\alpha_i \neq 0$, $i \in [M]$, can be any non-zero distinct real numbers. The next proposition quantifies the resilience of the MDS assignment to stragglers.

Proposition 3. *Under Assumption 4 with* $\mathbf{C}$ *in* (5.13)*, the probability that the distributed training procedure in Alg. 3 will be influenced by stragglers is* $\sum_{j=N-M+1}^{N} \binom{N}{j} (1-\eta)^{N-j} \eta^j$.

Proof. See Appendix B.3. □

Remark 4. *Under an MDS coding scheme* (5.13)*, each agent is assigned to all learners since all entries in the assignment matrix* $\mathbf{C}^{MDS}$ *are non-zero.*

Random Sparse Code

Compared to an MDS code, a Random Sparse code [212] results in sparser assignment matrices. The (j,i)-th entry of the assignment matrix, denoted $\mathbf{C}^{\text{Random}}$, is determined as follows:

$$\mathbf{C}_{j,i}^{\text{Random}} = \begin{cases} 0, & \text{with probability } 1-\xi, \\ \zeta, & \text{with probability } \xi. \end{cases} \tag{5.14}$$

where $\zeta \sim \mathcal{N}(0,1)$ and $\xi \in [0,1]$.

Remark 5. *Under a Random Sparse coding scheme* (5.14)*, the sparsity of the assignment matrix is determined by the parameter ζ. The smaller ζ is, the sparser $\mathbf{C}^{Random}$ is and the fewer learners each agent is assigned to.*

Repetition Code

A Repetition code [212] assigns agents to the learners repetitively in a round-robin fashion. The (j,i)-th entry of the assignment matrix under this scheme, denoted $\mathbf{C}^{\text{Repetition}}$, is given by:

$$\mathbf{C}_{j,i}^{\text{Repetition}} = \begin{cases} 1, & \text{if } i = (j \bmod M) + M\mathbb{1}_{(j \bmod M)=0}, \\ 0, & \text{else}, \end{cases} \tag{5.15}$$

where mod is the modulo operator and $\mathbb{1}$ is an indicator function.

Remark 6. *Under a Repetition coding scheme* (5.15)*, each agent is assigned to at least $\lfloor \frac{N}{M} \rfloor$ learners.*

LDPC Code

The assignment matrix of an LDPC code [213] is constructed using a parity check matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_\omega & \mathbf{I}_\omega & \mathbf{I}_\omega & \cdots & \mathbf{I}_\omega \\ \mathbf{I}_\omega & \mathbf{A} & \mathbf{A}^2 & \cdots & \mathbf{A}^{\frac{N}{\omega}-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{I}_\omega & \mathbf{A}^{\frac{Y}{\omega}-2} & \mathbf{A}^{2(\frac{Y}{\omega}-2)} & \cdots & \mathbf{A}^{(\frac{Y}{\omega}-2)(\frac{N}{\omega}-1)} \\ \mathbf{I}_\omega & \mathbf{A}^{\frac{Y}{\omega}-1} & \mathbf{A}^{2(\frac{Y}{\omega}-1)} & \cdots & \mathbf{A}^{(\frac{Y}{\omega}-1)(\frac{N}{\omega}-1)} \end{bmatrix} \in \mathscr{F}_2^{Y \times N},$$

where $\mathscr{F}_2$ denotes the binary field, ω is a prime number satisfying $\frac{N}{\omega} \in \mathbb{Z}^+$, $Y \in [N]$ satisfies $\frac{Y}{\omega} \in \mathbb{Z}^+$, $\mathbf{I}_\omega \in \mathbb{R}^{\omega \times \omega}$ is an identity matrix, and $\mathbf{A}$ is a permutation matrix given by:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \in \mathscr{F}_2^{\omega \times \omega}.$$

The LDPC assignment matrix $\mathbf{C}^{\text{LDPC}}$ can then be obtained by:

$$\mathbf{C}^{\text{LDPC}} = [\mathbf{I}_M | \mathbf{P}]^T \in \mathscr{F}_2^{N \times M}, \tag{5.16}$$

where $\mathbf{P}$ is obtained by solving $\mathbf{H} = [-\mathbf{P}^T | \mathbf{I}_{N-M}]$ with $M = N - \omega - (\frac{Y}{\omega} - 1)(\omega - 1)$.

LDGM Code

An LDGM code [214] is a special type of an LDPC code that constructs a sparser assignment matrix. By applying a systematic biased random code ensemble [214], the LDGM

assignment matrix takes the form:

$$\mathbf{C}^{\text{LDGM}} = [\mathbf{I}_M|\hat{\mathbf{P}}]^T \in \mathscr{F}_2^{N\times M}, \tag{5.17}$$

where each entry of $\hat{\mathbf{P}}$ is generated independently according to a Bernoulli distribution with success probability $\Pr(\hat{\mathbf{P}}_{i,j} = 1) = \rho$. Note that when $\rho \leq \frac{1}{2}$, the assignment matrix of LDGM code has a low density.

5.8 Experiments

In this section, we evaluate the DARL1N algorithm and our coding schemes for distributed training.

5.8.1 Performance of DARL1N

We conduct a series of comparisons between DARL1N and four state-of-the-art MARL algorithms. For fair comparison, we train DARL1N using a distributed learning architecture with uncoded assignments (5.12), and assume a reliable computing system without stragglers.

Experiment Settings

This subsection describes the experiment settings.

Environment Configurations

We evaluate DARL1N in five environments, including the Ising Model [185], Food Collection, Grassland, Adversarial Battle [186], and Multi-Access Wireless Communication [184], which cover cooperative and mixed cooperative competitive games. Please refer to [184, 185, 186] for the description of each environment.

To understand the scalability of our method, we vary the number of agents M and the size of the local state spaces. The number of agents in the Ising Model and Food Collection environments is set to $M = 9, 16, 25, 64$ and $M = 3, 6, 12, 24$, respectively. In the Grassland

and Adversarial Battle environments, the number of agents is set to $M = 6, 12, 24, 48$. In the Multi-Access Wireless Communication environment, we adopt the setting in [184] and consider a grid of 3×3 agents, with each having a state space of $\mathscr{S}_i = \{0,1\}^z$ to indicate whether there is a packet to send by time step z. In the experiments, z is set to either $z = 2$ or $z = 10$.

Distance Metrics

The one-hop neighbors of an agent are defined over the agent's state space using a distance metric. In the Ising Model and Multi-Access Wireless Communication environments, the topology of the agents is fixed, and the one-hop neighbors of an agent include its vertically and horizontally adjacent agents and itself. In the other environments, the Euclidean distance between two agents in 2-D space is used, and the neighbor distance d is set to 0.15, 0.2, 0.25, 0.3, 0.35 when $M = 3, 6, 12, 24, 48$, respectively. The bound ε for potential neighbors is determined according to the maximum velocity and time interval between two consecutive time steps, and is set to 0.05, 0.10, 0.15, 0.20, 0.25 when $M = 3, 6, 12, 24, 48$, respectively. The size of agents' activity space is set to $[-1,1] \times [-1,1]$, $[-1.5,1.5] \times [-1.5,1.5]$, $[-2,2] \times [-2,2]$, $[-2.5,2.5] \times [-2.5,2.5], [-3,3] \times [-3,3]$ when $M = 3, 6, 12, 24, 48$, respectively.

Benchmarks

We compare our method with four state-of-the-art MARL algorithms: MADDPG [178], MFAC [185], EPC [186], and SAC [184]. The SAC algorithm only works in the Multi-Access Wireless Communication environment due to the reward assumption and thus is not considered in other environments.

Evaluation Metrics

We measure the performance using two criteria: *training efficiency* and *policy quality*. To measure the training efficiency, we use two metrics: 1) *average training time* spent to run a specified number of training iterations and 2) *convergence time*. The convergence time is defined as the time when the variance of the average total training reward over 90 consecutive iterations does not exceed 2% of the absolute mean reward, where the average total training reward is the

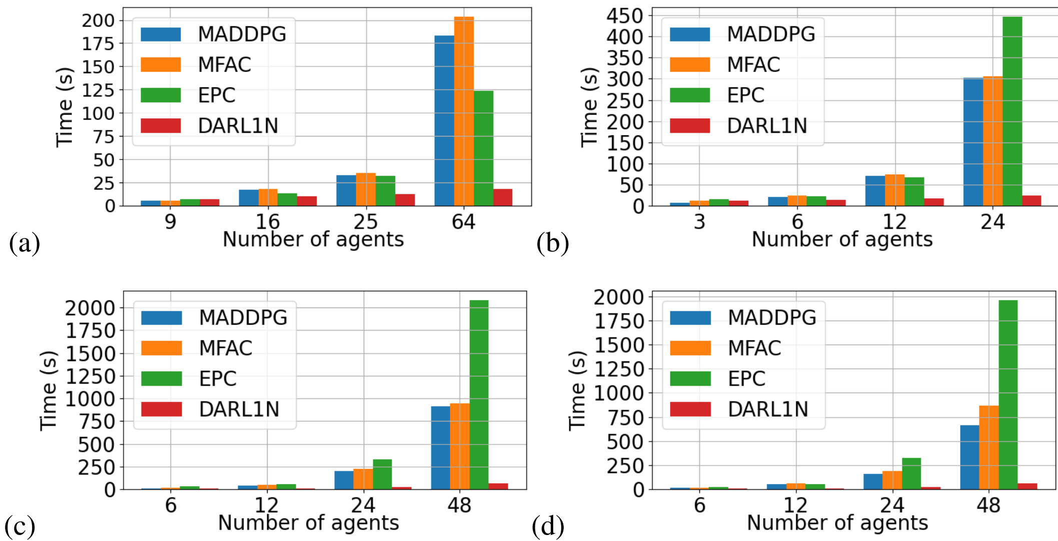


Figure 5.2. Average training time of different methods to run (a) 10 iterations in the Ising Model, (b) 30 iterations in the Food Collection, (c) 30 iterations in the Grassland, and (d) 30 iterations in the Adversarial Battle environments.

total reward of all agents averaged over 10 episodes in three training runs with different random seeds. To measure policy quality, we use *convergence reward*, which is the average total training reward at the convergence time.

Computing Configurations

We run our experiments on the Amazon EC2 computing clusters [131]. To evaluate the training efficiency, we configure the computing resources used to train each method in a way so that DARL1N utilizes roughly the same or fewer resources than the baseline methods. To train DARL1N, Amazon EC2 instance $c5n.large$ is used in all scenarios for all environments. To train MADDPG and MFAC, instance $z1d.3xlarge$ is used in the first scenario ($M = 9$) for Ising Model and in the first two scenarios for Food Collection, Grassland and Adversarial Battle, and in all scenarios for Multi-Access Wireless Communication. In the other scenarios, instance $z1d.6xlarge$ is used. To train EPC, we use instance $c5.12xlarge$ in all scenarios for Food Collection and in the first three scenarios for Ising Model, Grassland and Adversarial Battle. The other scenarios use instance $c5.18xlarge$. To configure the parallel computing architecture in EPC, we set the number of parallel computing instances and the number of independent

Table 5.1. Configurations of Amazon EC2 instances

Instances	CPU cores	CPU frequency	Memory	Network	Hourly price
c5n.large	2	3.4 GHz	5.3 GB	$\leq$ 25 Gb	$ 0.108
z1d.3xlarge	12	4 GHz	96 GB	$\leq$ 10 Gb	$ 1.116
z1d.6xlarge	24	4 GHz	192 GB	$\leq$ 10 Gb	$ 2.232
c5.12xlarge	48	3.6 GHz	96 GB	12 Gb	$ 2.04
c5.18xlarge	72	3.6 GHz	144 GB	25 Gb	$ 3.06

environments to 3 and 25, respectively. More configuration details including the configurations of the selected Amazon EC2 instances are provided in Tab. 5.1.

Training Parameters

All environments use the same training parameters. In particular, the Adam optimizer [215] is used to update the policy and Q function parameters with a learning rate of 0.01. The parameter τ in the Polyak averaging algorithm for updating the target policy and target Q functions is set to $\tau = 0.01$. The discount factor γ is set to $\gamma = 0.95$. The size of the replay buffer is set to 10^6. For mini-batches, the size is set to 32 in the Ising Model and 1024 in other environments. The parameters are updated after every 4 episodes. The *max_transition_number* in Alg. 3 of DARL1N is set to 4 times of the length of one episode. In the Ising Model and Food Collection environments, the length of each episode is set to 25 in all scenarios. In the Grassland and Adversarial Battle environments, the length of an episode is set to $25, 30, 35$ and 40 for the scenarios of $M = 6, 12, 24$ and 48, respectively.

Q Function and Policy Function Representations

For DARL1N, MADDPG, MFAC and SAC, we use neural networks with fully connected layers to represent the approximated Q function and policy function. The neural networks have three hidden layers with each layer having 64 units and adopting ReLU as the activation function. To handle the varying sizes of $\mathbf{s}_{\mathcal{N}_i}$ and $\mathbf{a}_{\mathcal{N}_i}$ in the approximated Q function in DARL1N, we let the input dimension of the approximated Q function to be the size of the joint state and action

Table 5.2. Convergence time and convergence reward of different methods in the Ising Model environment.

Method	Convergence Time (s)				Convergence Reward			
	$M=9$	16	25	64	9	16	25	64
MADDPG	62	263	810	1996	460	819	**1280**	1831
MFAC	63	274	851	2003	**468**	814	1276	1751
EPC	101	**26**	**51**	**62**	**468**	**831**	1278	**3321**
EPC Scratch	101	412	993	2995	**468**	826	1275	2503
DARL1N	**38**	102	210	110	465	828	1279	2282

space of the maximum number of agents that can be in $\mathcal{N}_i$. In particular, in the Ising Model, the maximum number of one-hop neighbors of an agent is 5, which is fixed. The input dimension of the approximated Q function for agent i is then $5 \times (|\mathcal{S}_i| + |\mathcal{A}_i|)$. For other environments, the maximum number of one-hop neighbors of an agent is the total number of agents. The EPC adopts a population-invariant neural network architecture with attention modules to support arbitrary number of agents in different stages for training the Q and policy functions.

Experiment Results

This subsection presents the main experiment results.

Ising Model

Tab. 5.2 shows the convergence reward and convergence time of different methods. When the number of agents is small ($M = 9$), all methods achieve roughly the same reward. DARL1N takes the least amount of time to converge while EPC takes the longest time. When the number of agents increases, it can be observed that the EPC converges immediately and the convergence reward it achieves when $M = 64$ is much higher than the other methods. The reason is that, in the Ising Model, each agent only needs information of its four fixed neighbors, and hence in EPC the policy obtained from the previous stage can be applied to the current stage. The other methods train the agents from scratch without curriculum learning. For illustration, we also show the convergence reward and convergence time achieved by training EPC from scratch

Table 5.3. Convergence time and convergence reward of different methods in the Food Collection environment.

Method	Convergence Time (s)				Convergence Reward			
	$M = 3$	6	12	24	3	6	12	24
MADDPG	**501**	1102	4883	2005	24	24	-112	-364
MFAC	512	832	4924	2013	20	23	-115	-362
EPC	1314	723	2900	8104	**31**	**34**	-16	-87
DARL1N	502	**382**	**310**	**1830**	14	25	**13**	**-61**

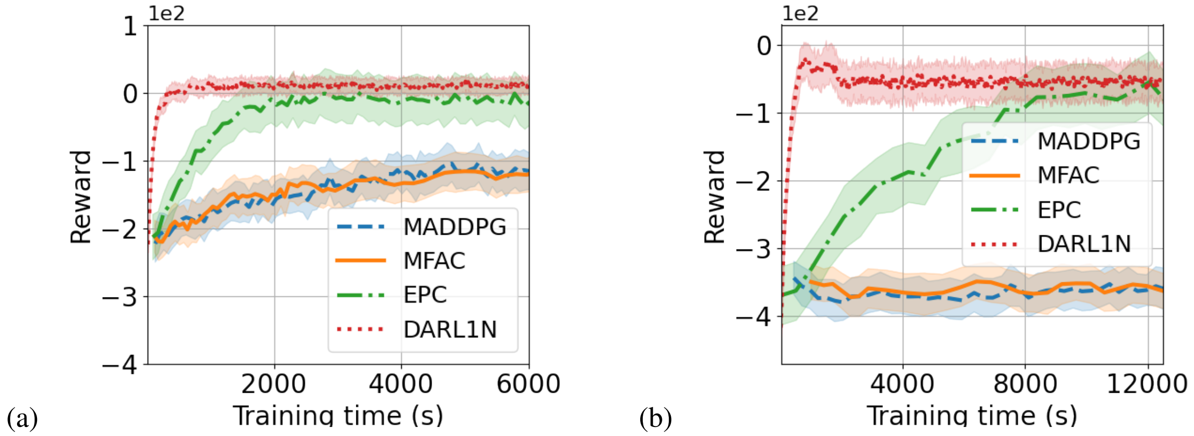


Figure 5.3. Average total training reward of different methods in the Food Collection environment when there are (a) $M = 12$, (b) $M = 24$ agents.

without curriculum learning (labeled as EPC Scratch in Tab. 5.2). The results show that EPC Scratch converges much slower than EPC as the number of agents increases. Note that when the number of agents is 9, EPC and EPC Scratch are the same. Moreover, DARL1N achieves a reward comparable with that of EPC Scratch but converges much faster. Fig. 5.2(a) shows the average time taken to train each method for 10 iterations in different scenarios. DARL1N requires much less time to perform a training iteration than the benchmark methods.

Food Collection

The convergence rewards and convergence times in this environment are shown in Tab. 5.3. The results show that, when the problem scale is small, DARL1N, MADDPG and MFAC achieve similar performance in terms of policy quality. As the problem scale increases,

the performance of MADDPG and MFAC degrades significantly and becomes much worse than DARL1N or EPC when $M = 12$ and $M = 24$, which is also illustrated in Fig. 5.3. The convergence reward achieved by DARL1N is comparable or sometimes higher than that achieved by EPC. Moreover, the convergence speed of DARL1N is the highest among all methods in all scenarios.

Fig. 5.2(b) shows the average training time for running 30 iterations. Similar as the results obtained in the Ising Model, DARL1N achieves the highest training efficiency and its training time grows linearly as the number of agents increases. When $M = 24$, EPC takes the longest time to train. This is because of the complex policy and Q neural network architectures in EPC, the input dimensions of which grow linearly and quadratically, respectively, with more agents.

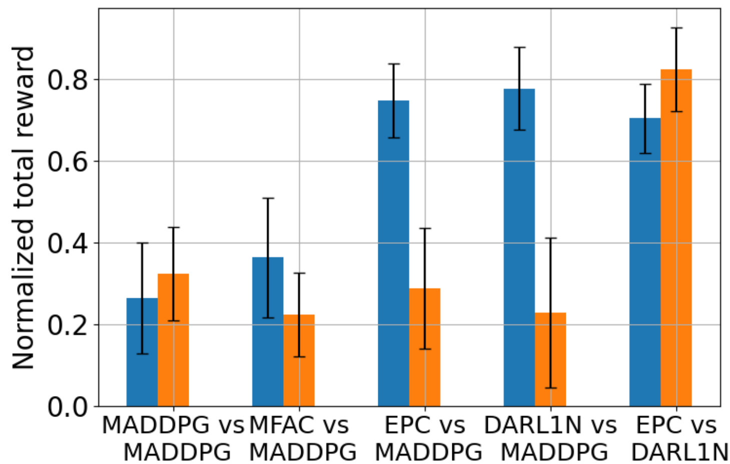


Figure 5.4. Mean and standard deviation of normalized total reward of competing agents trained by different methods in the Adversarial Battle environment with $M = 48$.

Grassland

Similar as the results in the Food Collection environment, the policy generated by DARL1N is equally good or even better than those generated by the benchmark methods, as shown in Tab. 5.4 and Fig. 5.2(c), especially when the problem scale is large. DARL1N also has the fastest convergence speed and takes the shortest time to run a training iteration.

Table 5.4. Convergence time and convergence reward of different methods in the Grassland environment.

Method	Convergence Time (s)				Convergence Reward			
	$M = 6$	12	24	48	6	12	24	48
MADDPG	423	6271	2827	1121	21	11	-302	-612
MFAC	431	7124	3156	**1025**	**23**	9	-311	-608
EPC	4883	2006	3324	15221	12	38	105	205
DARL1N	**103**	**402**	**1752**	5221	18	**46**	**113**	**210**

Table 5.5. Convergence time and convergence reward of different methods in the Adversarial Battle environment.

Method	Convergence Time (s)				Convergence Reward			
	$M = 6$	12	24	48	6	12	24	48
MADDPG	452	1331	1521	7600	-72	-211	-725	-1321
MFAC	463	1721	1624	6234	-73	-221	-694	-1201
EPC	1512	1432	2041	9210	-75	-215	**-405**	**-642**
DARL1N	**121**	**756**	**1123**	**3110**	**-71**	**-212**	-410	-682

Adversarial Battle

In this environment, DARL1N again achieves good performance in terms of policy quality and training efficiency compared to the benchmark methods, as shown in Tab. 5.5 and Fig. 5.2(d). To further evaluate the performance, we reconsider the last scenario ($M = 48$) and train the good agents and adversary agents using two different methods. The trained good agents and adversarial agents then compete with each other. We apply the Min-Max normalization to measure the normalized total reward of agents at each side achieved in an episode. To reduce uncertainty, we generate 10 episodes and record the mean values and standard deviations. As shown in Fig. 5.4, DARL1N achieves the best performance, and both DARL1N and EPC significantly outperform MADDPG and MFAC.

Multi-Access Wireless Communication

Fig. 5.6 shows the training rewards achieved by DARL1N and SAC when z takes different values. We can see that SAC achieves a higher reward than DARL1N when $z = 2$. However,

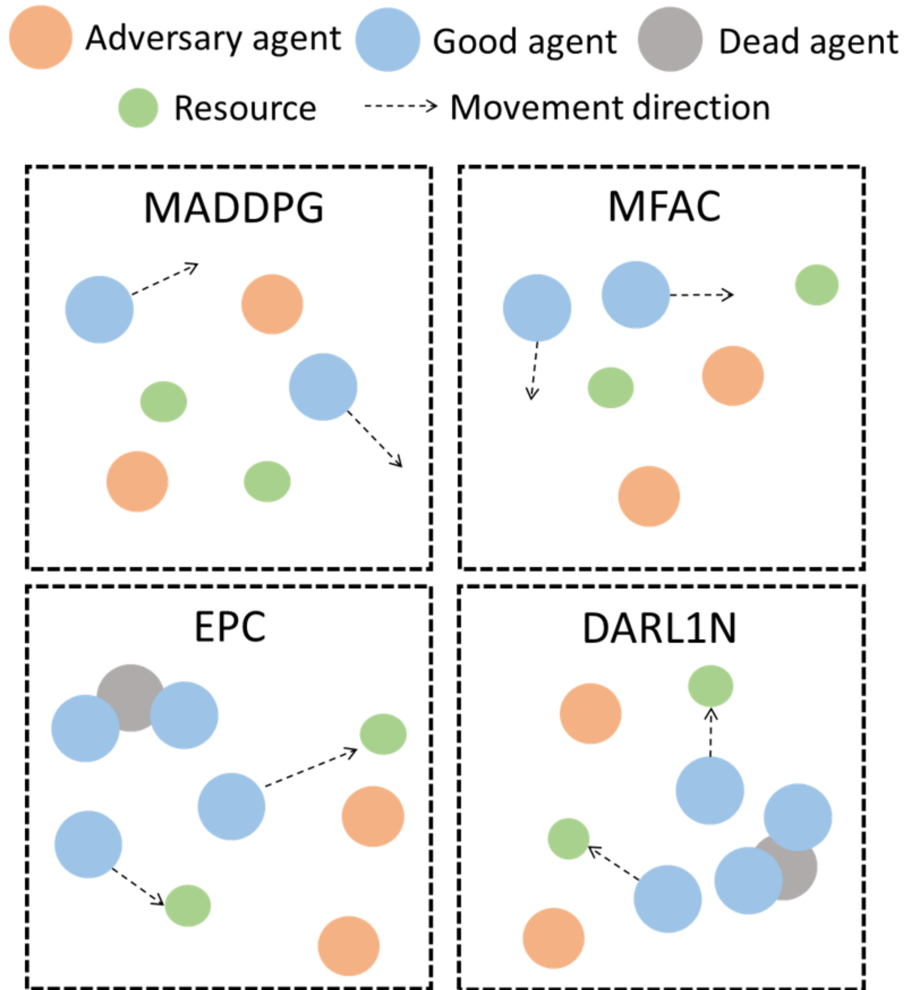


Figure 5.5. States of a subset of agents during an episode in Adversarial Battle with agents trained by different methods when there are $M = 48$ agents.

when z increases to 10, which causes an exponential growth of the state space, DARL1N achieves a much higher reward and converges much faster than SAC. This demonstrates that DARL1N scales better than SAC with the size of the state space.

5.8.2 Performance of Coded Distributed Learning Architecture

In this section, we first conduct numerical studies to evaluate the performance of different agent assignment schemes described in Sec. 5.7 before implementing DARL1N. We then train DARL1N over the proposed coded distributed computing architecture and conduct experimental studies to evaluate the performance of DARL1N when trained with different agent assignment

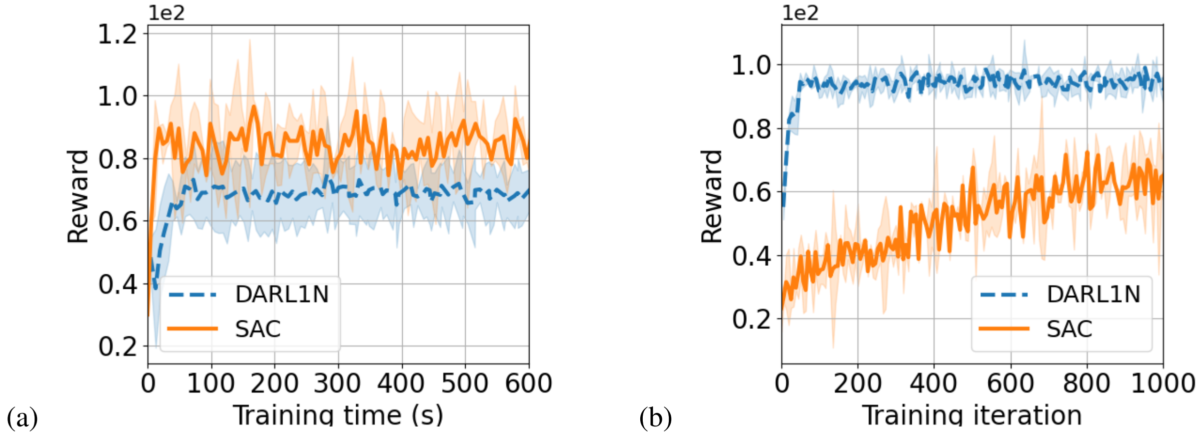


Figure 5.6. Average total training reward of DARL1N and SAC in the Multi-Access Wireless Communication environment when (a) $z = 2$ (b) $z = 10$.

schemes.

Numerical Evaluation

We first conduct numeral simulations to evaluate the performance of different agent assignment schemes based on the following aspects: 1) computation overhead, 2) resilience to stragglers, and 3) impact on policy quality.

Computation Overhead

The coded schemes mitigate the impact of stragglers on the distributed learning system by assigning each agent to more than one learner. The training performed by the extra learners is redundant. To measure the amount of redundant computation *overhead* introduced by each scheme, we use the following metric:

$$o = \frac{1}{M} \sum_{j=1}^{N} \sum_{i=1}^{M} \mathbb{1}_{\mathbf{C}_{j,i} \neq 0} - 1,$$

where the first term calculates the average number of learners used for training each agent.

Fig. 5.7 shows the overhead introduced by different schemes when $M = 12$, $N = 24$, $\rho = 0.3$, and $\xi = 0.8$. For Random Sparse and LDGM schemes, the mean overhead averaged

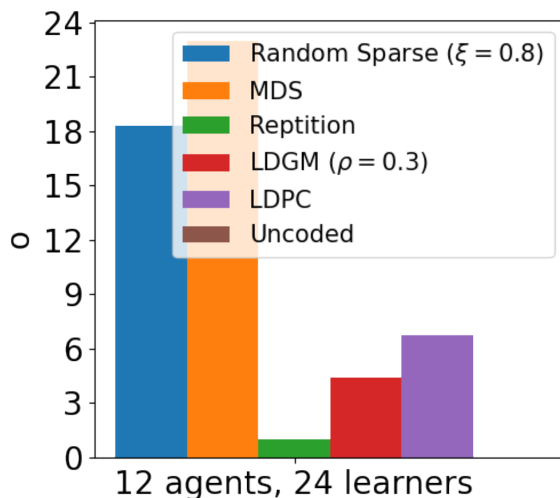


Figure 5.7. Overhead introduced by different agent assignment schemes when there are $M = 12$ agents and $N = 24$ learners.

over 10 experiment runs are shown. We can see that the Random Sparse and MDS schemes introduce larger overhead than the other four schemes. The MDS scheme generates the largest overhead, as it assigns each agent to all learners. On the contrary, as the uncoded scheme assigns each agent to a single learner, it does not introduce any redundant computation. Generally, we can observe that schemes generating denser assignment matrices introduce larger overhead.

As the Random Sparse and LDGM schemes are characterized by the parameters ξ and ρ, respectively, in Fig. 5.8 we vary the values of these parameters to understand their impact on the overhead. The increase of ξ or ρ leads to denser assignment matrices and we can see that the overhead increases too.

Resilience to Stragglers

According to (5.8), the central controller is able to update the agents' gradients only after it receives results from enough learners, specifically when $\text{rank}(\mathbf{C}_{\mathscr{I}}) = M$. The presence of stragglers may delay or fail the gradient computations. To evaluate the resilience of different coding schemes to stragglers, we randomly turn some learners into stragglers that fail to return any results according to Assumption 4. We then conduct Monte Carlo simulations to measure the ratio of training iterations in which gradients can be successfully estimated with results returned

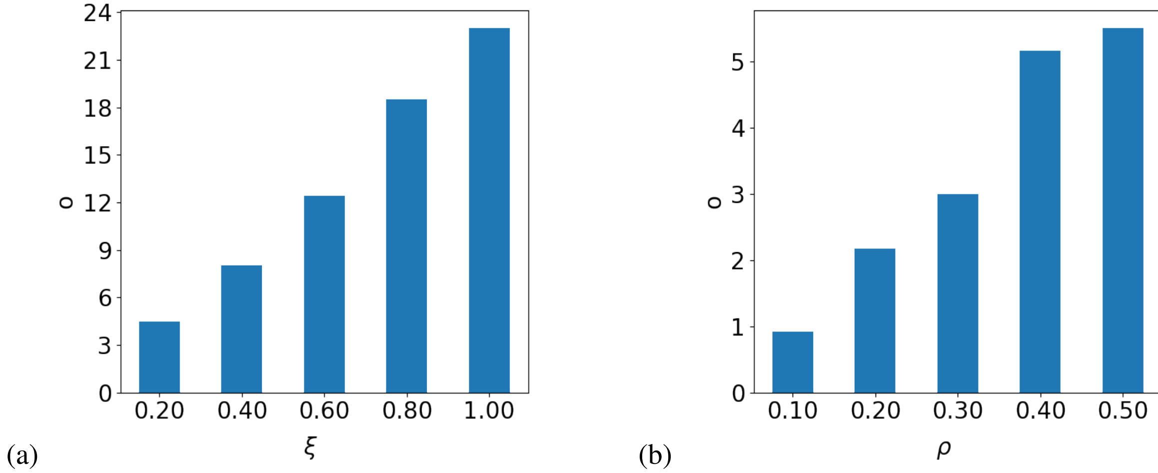


Figure 5.8. Overhead introduced by (a) Random Sparse and (b) LDGM schemes when their parameters take different values.

from non-stragglers. Fig. 5.9 shows the ratio of successful iterations achieved by different schemes as the straggler probability η (chance of becoming a straggler for each learner at each iteration) increases when $M = 12$ and $N = 24$. The Random Sparse and MDS schemes achieve the highest resilience and can tolerate up to 12 stragglers, as their assignment matrices are the densest. LDPC is more robust than LDGM and Repetition based schemes. The Uncoded scheme performs the worst and cannot tolerate any stragglers. These results demonstrate that higher resilience can be achieved if using schemes that generate a denser assignment matrix.

The resilience of the LDGM and Random Sparse schemes are also affected by their parameters, ρ and ξ, respectively. From Fig. 5.10, we can see that with the increase of ρ and ξ, both schemes can tolerate more stragglers due to the growing density of the resulting assignment matrix.

Impact on Policy Quality

To understand the impact of different coding schemes on the quality of trained policies, we use the following metric that reflects the variance of the estimated gradients:

$$V = \log(\det(\mathrm{Var}[\hat{\mathbf{e}}])). \tag{5.18}$$

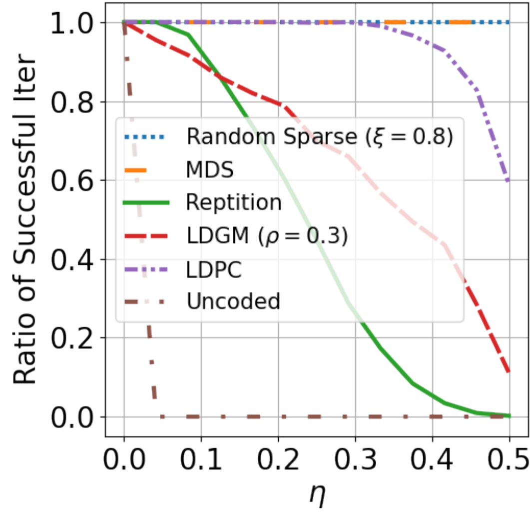


Figure 5.9. Resilience of different agent assignment schemes to stragglers when the straggler probability η increases.

To calculate the value of V, we set $\mathrm{Var}(\hat{\mathbf{e}}_{i,j}) = 1, \forall i \in [N], j \in [M]$, and vary the number of learners whose results are utilized by the central controller for estimating the gradients. The results are shown in Fig. 5.11, where each value is an average over 20 experiment runs. We can see that V decreases as more learners contribute to the estimation of the gradients. Moreover, the Repetition code generates the smallest V, indicating that it has the least impact on the policy quality. The Random Sparse, LDPC, and LDGM schemes achieve a similar performance with V close to 0. On the contrary, the value of V obtained using an MDS scheme is relatively high, indicating large variance of the estimated gradients, which may lead to low-quality learned policies.

Experiments

To understand the performance of the coded distributed learning architecture, we train DARL1N using different agent assignment schemes and evaluate its performance in different straggler scenarios.

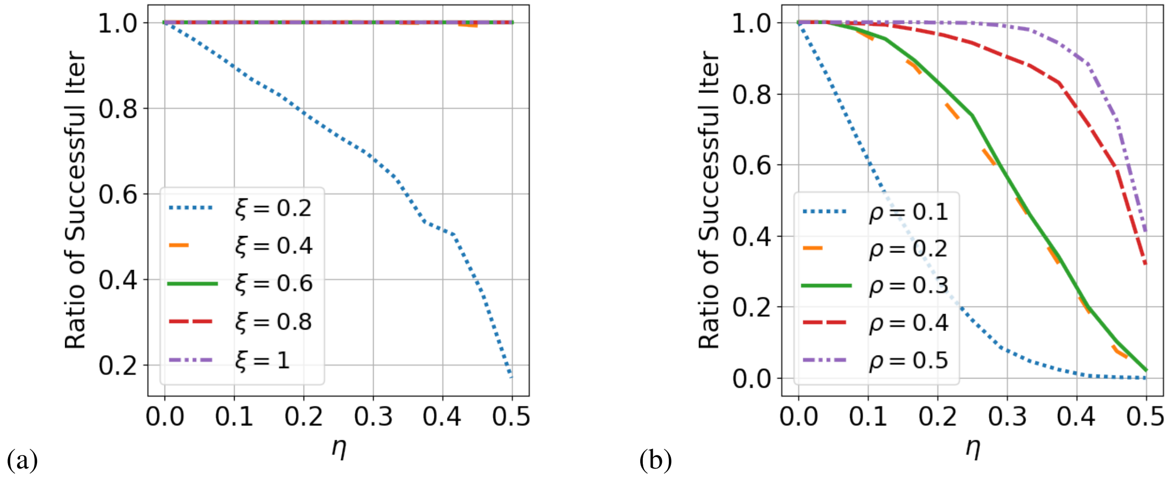


Figure 5.10. Resilience of (a) Random Sparse and (b) LDGM schemes when their parameters take different values.

Experiment Settings

We select the Food Collection environment and set the number of agents and learners in all experiments to $M = 12$ and $N = 24$, respectively. To evaluate the impact of stragglers, we vary the straggler probability η in Assumption 4. The straggler effect is simulated by letting the stragglers delay returning results for $\Delta > 0$ amount of time using the $sleep()$ function.

Experiment Results

We first evaluate the average training time of DARL1N with different agent assignment schemes. We vary the straggler probability η and the straggler effect Δ. The results are shown in Fig. 5.12(a) when $\Delta = 1$ and Fig. 5.12(b) when $\Delta = 4$, where the training time is measured by averaging the time for running 30 training iterations. We can observe that when no stragglers exist ($\eta = 0$), the Uncoded scheme is the most efficient as it does not involve any redundant computations. The MDS and Random Sparse schemes require a much longer training time than the other schemes because of the many redundant computations introduced by these codes. When stragglers exist ($\eta > 0$), the performance of the Uncoded scheme degrades significantly, especially when the straggler effect is significant as shown in Fig. 5.12(b). Compared to the Uncoded scheme, the LDPC, LDGM, and Repetition codes are more resilient to stragglers, as

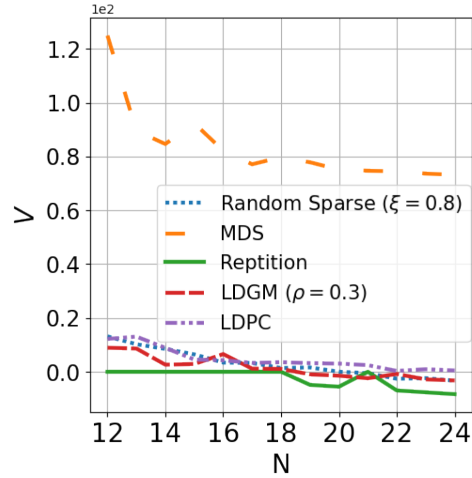


Figure 5.11. Average V of different agent assignment schemes calculated using results from different number of learners.

indicated by the slower increase in training time. They are also more efficient than the MDS and Random Sparse schemes in most cases. On the contrary, the training time of MDS and Random Sparse does not grow much as η increases, evidencing their high resilience to stragglers. Although they require more training time than the other schemes when the straggler effect Δ or the straggler probability η is small, they achieve higher training efficiency when Δ and/or η are large.

To evaluate the impact of different agent assignment schemes on the quality of trained policies, we measure the training reward achieved by each DARL1N implementation with $\Delta = 1$. The convergence time and convergence reward of different implementations as the straggler probability η increases are summarized in Tab. 5.6. We can see that the Uncoded scheme converges fast when no stragglers exist ($\eta = 0$) but its convergence speed decreases significantly when stragglers exist ($\eta > 0$). The MDS and Random Sparse schemes achieve the lowest reward and slowest convergence rate, while the LDPC and LDGM based implementations achieve the highest reward and convergence rate in most cases, especially when the straggler probability η is high. The Repetition code generally achieves good training reward performance and converges fast when the straggler probability is small. From these results, we can infer that a sparser

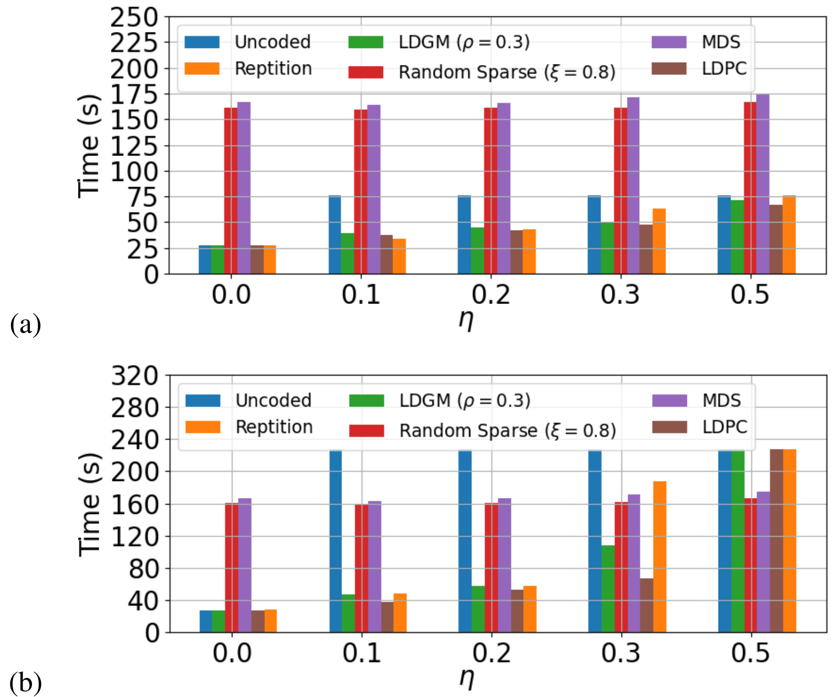


Figure 5.12. Average training time of different DARL1N implementations with straggler effect a) $\Delta = 1$ and b) $\Delta = 4$.

assignment matrix generally leads to a better policy and a higher convergence rate.

We also calculate the variance of the gradients estimated by each DARL1N implementation, measured by V in (5.18). Tab. 5.7 summarizes the values of V averaged over 600 training iterations for different implementations. We can see that the MDS and Random Sparse schemes have relatively higher V, indicating worse training reward performance. The variance V of the Repetition scheme is the smallest, while the variance of other implementations are close to zero regardless of the increase of η, which matches with the numerical results shown in Fig. 5.11. Of interest, V obtained by the LDGM and Repetition schemes are even smaller than that of the Uncoded implementation in most cases. This is because the two coded implementations may use results from more than one learner to estimate the gradients for each agent, while the Uncoded scheme uses results from a single learner only.

Table 5.6. Convergence time and convergence reward of different DARL1N implementations.

Schemes	Convergence Time (s)					Convergence Reward				
	$\eta = 0$	0.1	0.2	0.3	0.5	0	0.1	0.2	0.3	0.5
Uncoded	323	502	510	532	521	8	9	5	**13**	10
MDS	502	752	748	771	625	-231	-202	-253	-255	-212
Random Sparse	512	1252	820	620	670	-252	-255	-231	-241	-227
Repetition	331	**248**	372	605	564	11	7	6	4	8
LDPC	**318**	253	364	**310**	**420**	**12**	**11**	8	5	9
LDGM	324	261	**320**	331	450	9	10	**12**	7	**14**

Table 5.7. Average V of different DARL1N implementations.

Schemes	Average V				
	$\eta = 0$	0.1	0.2	0.3	0.5
Uncoded	0	0	0	0	0
MDS	82.08	85.75	100.10	103.25	104.06
Random Sparse	15.28	13.11	13.47	13.69	12.88
Repetition	**-3.16**	**-3.27**	**-4.49**	**-4.36**	**-4.15**
LDPC	0	0.14	0.13	0.03	0.03
LDGM	-0.66	-0.61	-0.51	0.14	0.35

5.9 Conclusion

This chapter introduced DARL1N, a scalable MARL algorithm that can be trained over a distribute computing architecture. DARL1N reduces the representation complexity of the value and policy functions of each agent in a MARL problem by disregarding the influence of other agents that are not within one hop of a proximity graph. This model enables highly efficient distributed training, in which a compute node only needs data from an agent it is training and its potential one-hop neighbors. We conducted comprehensive experiments using five MARL environments and compared DARL1N with four state-of-the-art MARL algorithms. DARL1N generates equally good or even better policies in almost all scenarios with significantly higher training efficiency than benchmark methods, especially in large-scale problem settings. To improve the resilience of DARL1N to stragglers common in distributed computing systems, we developed coding schemes that assign each agent to multiple learners. The properties of

MDS, Random Sparse, Repetition, LDPC, and LDGM codes were evaluated. Our results show that the MDS and Random Sparse codes offer high resilience to large numbers of stragglers. However, both schemes involve many redundant computations and have low training efficiency. The Repetition, LDGM, and LDPC coding schemes achieve better performance in both training efficiency and policy quality but can handle fewer stragglers in the system.

5.10 Acknowledgement

This chapter is based on the published conference proceedings: B. Wang, J. Xie, N. Atanasov, "DARL1N: Distributed multi-Agent Reinforcement Learning with One-hop Neighbors", *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*; B. Wang, J. Xie, N. Atanasov, "Coding for Distributed Multi-Agent Reinforcement Learning", *2021 International Conference on Robotics and Automation (ICRA)*, and on the submitted journal paper: B. Wang, J. Xie, N. Atanasov, "Coding for Distributed multi-Agent Reinforcement Learning with One-hop Neighbors", *IEEE Transactions on Neural Networks and Learning Systems*, Nov. 2022.

Chapter 6

Simulator and Testbed Design and Implementation

6.1 Introduction

Before deploying the approaches introduced in previous chapters in actual UAV systems, it is imperative to test their performance on realistic testbeds. In Chapter 2, we designed an airborne computing platform that is equipped with NVIDIA Jetson TX2 of high computing power and also supports broadband wireless communication that can facilitate network-based UAV research. Nevertheless, its design is centered on a single UAV, and inter-vehicle resource sharing is not supported. Moreover, its communication module, equipped with a directional antenna, was designed and tested only for point-to-point communications. Nevertheless, NAC may involve point-to-multipoint or multipoint-to-multipoint communications. In this chapter, we aim to

In existing studies on UAV-based computing [19, 20, 21, 22, 23], performance evaluation was typically conducted through simulations with UAV movement, communication, and computing behaviors described using mathematical models. While mathematical model-based simulations offer the advantage of being inexpensive and easy to deploy, their underlying models, due to simplicity, may not accurately reflect the intricate behavior of real UAV systems. Hardware tests have been limited to single or dual UAVs. For example, [216] investigated the application of airborne computing for computer vision and machine learning, and developed a

UAV-based MEC system with two UAVs to analyze communication and computation delays for video streaming between the two UAVs.

Although the absence of realistic testbeds for NAC research remains, various testbeds have been created to aid research in UAV control, communication, and networking [217, 218, 219, 220, 221, 222, 223]. For instance, [217] created a testbed to investigate the communication and control in UAV swarms. [218] introduced a UAV hardware testbed for control and information acquisition. To support robotics applications, such as collision avoidance, [219] built a UAV simulator that runs on the cloud using Docker. In the realm of UAV communication, [220] presented a new evolved packet core (EPC) that can be placed directly on UAV to establish UAV-based LTE networks. They evaluated its performance in a two-UAV LTE network, which resulted in improved client connectivity. [221] developed a directional antenna-based broadband and long-range communication system for UAV-to-UAV communication, which has the ability to automatically adjust antenna directions for optimizing communication performance in unknown communication environments.[222] proposed a new UAV network design using mmWave for Gigabit speed communication and evaluated it on a hardware testbed. Additionally, [223] created a UAV simulator to investigate the communication security issue.

Main contributions: In this chapter, we aim to address the research gap regarding the scarcity of realistic testbeds for NAC research. To this end, two NAC platforms are designed and implemented. One is a realistic simulator created using ROS (Robot Operating System) [26] and Gazebo [27]. Another is a hardware testbed composed by multiple UAVs with computing and inter-vehicle resource sharing capabilities. Moreover, we conduct various simulations and real flight tests with two computation applications to examine the impact of UAV mobility on NAC. The obtained insights are vital for progressing NAC research by uncovering the barriers to achieving high-performance NAC. They also provide guidance for future enhancements of the proposed NAC platforms.

The rest of the chapter is organized as follows. Sec. 6.2 presents the computing model for UAV-based NAC. Sec. 6.3 and Sec. 6.4 describe the designs for the simulator and hardware

Algorithm 4: Computing model in NAC

```
  // Master:
1 Broadcast X to workers.
2 for i = 1 : N do
3 └ Listen to the channel and collect result f_i(X) from worker i.
4 Calculate and output final result using {f_1(X), f_2(X), ..., f_N(X)}
  // Worker i:
5 Listen to the channel and receive X from the master.
6 Compute f_i(X) and send results back to the master.
```

testbed, respectively. The simulation and flight test results are presented in Sec. 6.5 and Sec. 6.6, respectively. Finally, Sec. 6.7 concludes the chapter.

6.2 Computing Model

In this section, we describe how a computation task is completed collaboratively by UAVs in NAC. Consider a NAC system with $N+1$ UAVs, where N UAVs are the workers and one UAV serves as the master. Suppose the master UAV needs to complete a decomposable computation task $\mathbf{f}(\mathbf{X})$, where $\mathbf{X} \in \mathbb{R}^{m \times n}$ is the input to the task. To enhance computational efficiency, the master partitions the task into N subtasks represented as $\{f_1(\mathbf{X}), f_2(\mathbf{X}), \ldots, f_N(\mathbf{X}))\}$, and assigns each worker i with subtask $f_i(\mathbf{X})$. During execution, the master shares the input $\mathbf{X}$ with all the workers, which then calculate their respective subtasks and send the results back to the master. The master then combines the results to produce the final output of the task $\mathbf{f}(\mathbf{X})$. Algorithm 4 summarizes the task execution procedure.

In subsequent simulation and experimental studies, we select two computation tasks to be executed using the above computing model. They are 1) matrix multiplication and 2) linear regression, both of which are fundamental components for more complicated computations such as those in machine learning applications [224, 225]. Here we briefly describe each of the tasks.

To perform a matrix multiplication task $\mathbf{AB}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a pre-stored matrix and $\mathbf{B} \in \mathbb{R}^{n \times l}$ is the input, the master equally divides $\mathbf{A}$ row-wise into N submatrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N$.

Suppose each submatrix $\mathbf{A}_i$ is pre-stored in the corresponding worker i. The master distributes $\mathbf{B}$ to each worker, which then computes $\mathbf{A}_1\mathbf{B}, \mathbf{A}_2\mathbf{B}, \ldots, \mathbf{A}_N\mathbf{B}$, respectively, and returns the results to the master. The master then aggregates the received results $[\mathbf{A}_1\mathbf{B}, \mathbf{A}_2\mathbf{B}, \ldots, \mathbf{A}_N\mathbf{B}]$ to obtain the final result $\mathbf{AB}$.

For linear regression, we use a real-world data set collected in Montesinho park [226] as input to train a forest fire detector. The goal is to predict burned areas given meteorological information by training a linear regressor using gradient descent. The data set consists of 517 meteorological data samples, each containing information such as location in the park, time, temperature, humidity, wind speed, and the burned area of the forest. To train the linear regressor in NAC, the data set is divided into N equal parts, with each part stored in a worker. The master starts by initializing the parameters of the linear regressor and updating them iteratively. During each iteration, the master sends the current parameters to each worker, which calculates the gradients using their local data and returns the gradients to the master. The master then updates the parameters using the received gradients.

6.3 Simulator Design

This section presents a ROS- and Gazebo-based simulator designed to provide a realistic simulation environment for NAC research. The simulator (see Fig. 6.1) consists of five modules: UAV hardware module, controller module, visualization module, wireless communication module, and computation module. These modules communicate through ROS topics [227]. The functionalities of each module and their associated ROS topics are described as follows.

6.3.1 UAV Hardware Module

The UAV hardware module simulates the physical attributes of a UAV including its frame, motor, and sensors. Users can configure UAV's frame such as base size, weight, shape, arm length, etc. using the configuration file. Additionally, users can select different types of motors to simulate UAVs of different sizes and aerodynamics.

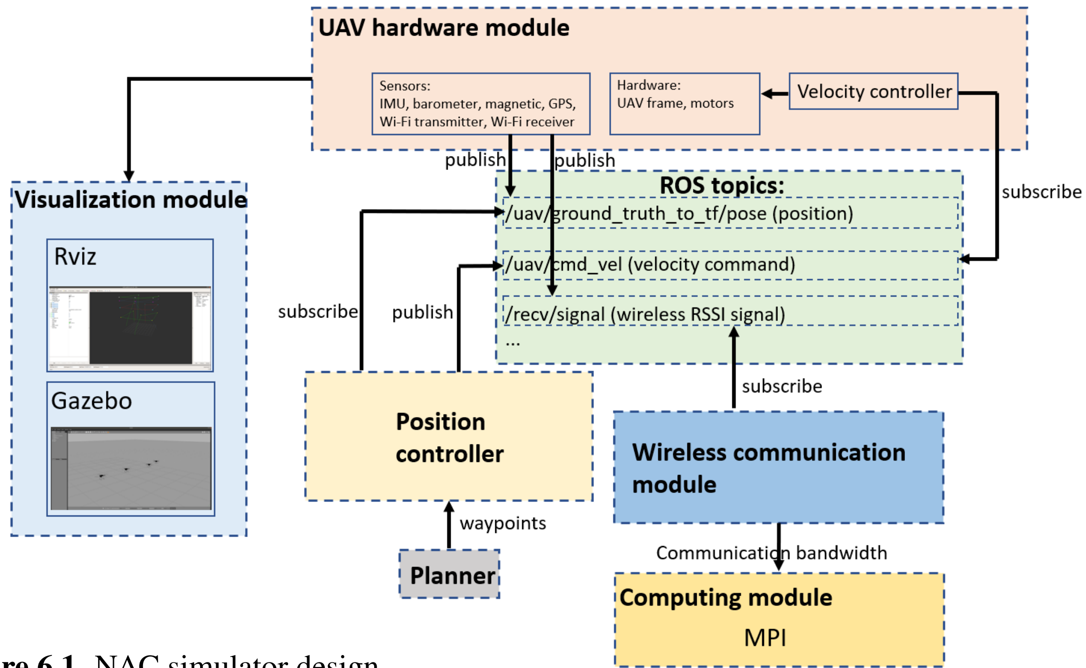


Figure 6.1. NAC simulator design.

ROS offers plugins for various sensors that can be attached to UAV to track its states, such as position, velocity, and orientation. In our simulator, each UAV is equipped with a GPS sensor for localization, a magnetic sensor for orientation, an IMU sensor for acceleration, and a barometer sensor for flight height measurement, etc. Moreover, each UAV includes a Wi-Fi receiver and Wi-Fi transmitter [228] to simulate wireless communication between UAVs. Users can also add additional sensors as needed.

6.3.2 Controller Module

The controller module controls UAV's movement to follow the planned waypoints. This is achieved by using two controllers: velocity controller and position controller. Particularly, the position controller calculates the desired velocities based on the planned waypoints and inputs the desired velocities into the velocity controller. The velocity controller, provided by ROS plugin, then generates control commands to drive the UAV to reach the desired velocities. The

calculation of the desired velocities is performed using the following equations:

$$\hat{v}_x = k_x(x - x_g)$$
$$\hat{v}_y = k_y(y - y_g)$$
$$\hat{v}_z = k_z(z - z_g) \tag{6.1}$$

where $\hat{v}_x, \hat{v}_y, \hat{v}_z$ are the desired velocities in three directions. k_x, k_y, k_z are the gains of the position controller. (x, y, z) and (x_g, y_g, z_g) are UAV's current position and the desired position, respectively.

6.3.3 Visualization Module

The visualization module allows users to monitor the status of UAVs during NAC simulations in real time. This is achieved using Gazebo [27], which provides a realistic simulation environment and an interactive graphic interface. Moreover, by using Rviz [229], the data captured by UAV's sensors, such as trajectories and images captured by the onboard camera, can be displayed. Our simulator provides the real-time visualization of waypoints and UAV trajectories.

6.3.4 Wireless Communication Module

The communication module simulates the wireless communication between UAVs, each equipped with a Wi-Fi transmitter and receiver. The received signal strength (RSS) at the Wi-Fi receiver, denoted by $S(dBm)$, is calculated using the well-known Hata-Okumura model [230, 231] by the following equation:

$$S = S_t + P_L, \tag{6.2}$$

where $S_t(dBm)$ is the transmission power and $P_L(dB)$ is path loss given by

$$P_L = D + E\log_{10}(d) + F. \tag{6.3}$$

In the above equation, $d(km)$ is the distance between two UAVs. The values of D, E, and F depend on the transmission frequency, antenna heights, and environment types. In particular, D and E are represented by

$$\begin{aligned} D &= 69.55 + 26.16log_{10}(f_c) - 13.82log_{10}(h_b) - a(h_m) \\ E &= 44.9 - 6.55log_{10}(h_b) \end{aligned} \tag{6.4}$$

where f_c is the frequency of transmission in MHz (valid range: $150MHz - 1500MHz$), h_b is the effective height of the transmitter in meters (valid range: $30m - 200m$), h_m is the effective height of the receiver (valid range: $1m - 10m$), and $a(h_m)$ is the mobile antenna height correction factor, which is a function that depends on the environment. F is a factor to correct formulas for open rural and suburban areas. Given RSS S, the maximum data rate, denoted by C (bps), can then be calculated according to Shannon's Theory:

$$C = W\log_2(1 + \frac{10^{\frac{(S-30)}{10}}}{N_0}), \tag{6.5}$$

where W (Hz) is the communication bandwidth and N_0 (Watts) is the noise power.

6.3.5 Computing Module

The computing module simulates the distributed computing process using Message Passing Interface (MPI)[232], which provides system interfaces for users to program parallel computing applications and supports various programming languages. In our simulator, each process represents a UAV that runs computation tasks in parallel and communicates with each other via MPI. Each process uses the states of UAV to calculate the data rate using (6.5). We

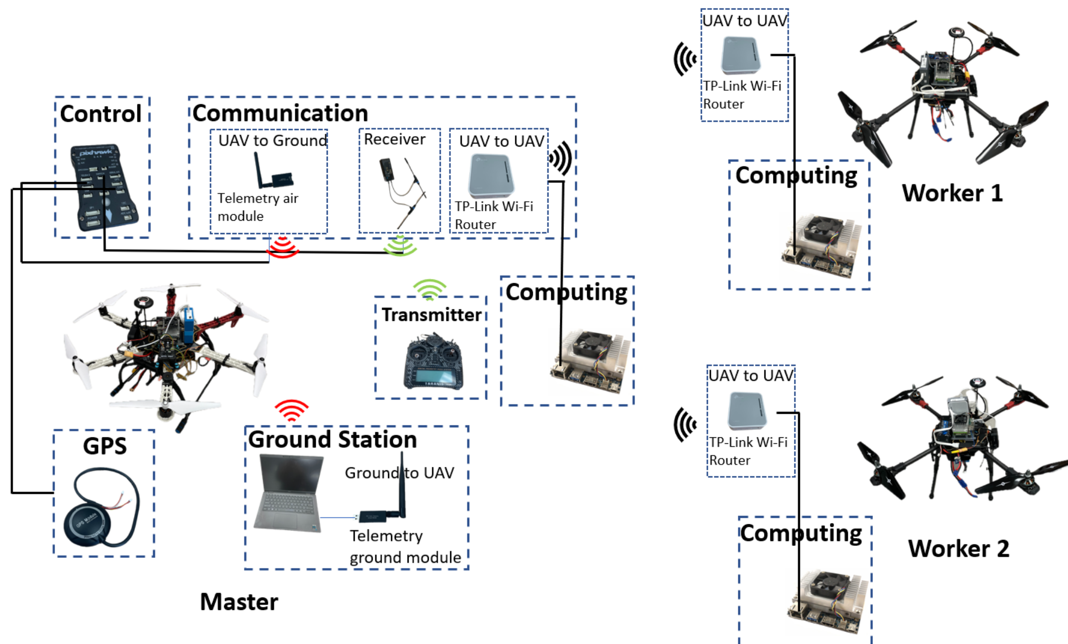


Figure 6.2. NAC hardware testbed design.

calculate the delay as $\frac{I}{C}$ (s), where $I(bits)$ is the size of data being transmitted. This delay is introduced manually using the $time.sleep()$ function in Python before sending data via MPI.

6.3.6 ROS Topics

In our simulator, communication between modules is achieved through ROS topics as shown in Fig. 6.1. Each module can either subscribe to a topic to receive messages or publish messages to a topic. For example, GPS publishes UAV's position to the topic $/uav/ground_truth_to_tf/pose$. The position controller subscribes to this topic to receive the UAV's current position. After calculating the desired velocities, the position controller then publishes these values to the topic $/uav/cmd_vel$, which is subscribed by the UAV's velocity controller. Moreover, the Wi-Fi receiver publishes the RSS signals to the topic $/uav/recv_signal$, which is subscribed by the communication module to calculate the data rate.

6.4 Hardware Testbed Design

In this section, we present the design of a NAC hardware testbed, consisting of three UAVs, two constructed using Tarot 650 Quadrotor frames and one using a DJI F550 frame (see Fig. 6.2). The Tarot quadrotors serve as workers, while the F550 serves as the master. In the following, we describe the hardware design from four aspects: computing, communication, flight control, and power management. For each aspect, we detail its functionalities and the hardware devices used to achieve them.

6.4.1 Computing

The computing unit on each UAV is a NVIDIA Jetson TX2 [233], which boasts a powerful 6-core CPU and 256-core NVIDIA Pascal GPU, and has 8GB memory. Despite its compact size (50mm$\times$87mm) and lightweight (85g), Jetson TX2 requires a development board to operate and the original one from NVIDIA is too large (17cm$\times$17cm) to be placed on the UAV. To overcome this, we utilize a compact carrier board developed in our previous works [136]. To enable computing resource sharing among UAVs, we connect Jetson TX2 to the same Wi-Fi network as described in the following subsection. This allows UAVs to perform collaborative computing using MPI.

6.4.2 Communication

As illustrated in Fig. 6.2, there are three types of communications in NAC including UAV-to-UAV, UAV-to-Ground/Ground-to-UAV, and Transmitter-to-Receiver. The features and enabling techniques for each communication type are described as follows.

UAV-to-UAV Communication

The UAV-to-UAV communication is used to exchange data between UAVs, such as data needed for computation and UAV's state information. Our testbed achieves this through a TP-Link Wi-Fi router network. In particular, one TP-Link Wi-Fi router is set up as an access

point on the master while the remaining TP-Link Wi-Fi routers on the workers are configured as clients connecting to the master. This allows the master to communicate with both workers simultaneously. Each router is connected to the Jetson TX2 via an Ethernet port.

UAV-to-Ground/Ground-to-UAV Communication

The UAV-to-Ground/Ground-to-UAV communication is used to monitor the states of UAV such as its position, rotation, battery status, etc. It is enabled by the FPVDrone 500MW Radio Telemetry Kit 915 Mhz Air and the Ground data transmit module. The air data transmit module is connected to the Pixhawk telemetry port and the ground module is connected to a laptop USB port. In our design, we use the QGroundControl software [234] installed on the ground station, which is a laptop, to interact with the UAV such as commanding it to take off or land. We can also create different missions such as waypoint following and upload them to the flight controller.

Transmitter-to-Receiver Communication

The transmitter-to-receiver communication is used to control the UAV. The transmitter has joysticks that allow manual control of throttle, yaw, pitch, and roll angles. The UAV's flight modes can also be changed by adjusting the switches, such as the Manual control mode, Mission mode for completing pre-programmed missions, and Hold mode for maintaining altitude.

6.4.3 Flight Control

Each UAV is equipped with a Pixhawk flight controller, which is capable of controlling the UAV based on manual inputs or following planned waypoints with the aid of GPS signals. Both the receiver and the telemetry air module are connected to the Pixhawk.

6.4.4 Power Management

Each UAV is equipped with two 4S Lipo batteries with each battery having 14.8 Voltage to provide power to all its subsystems. The first battery is designated to power the computing

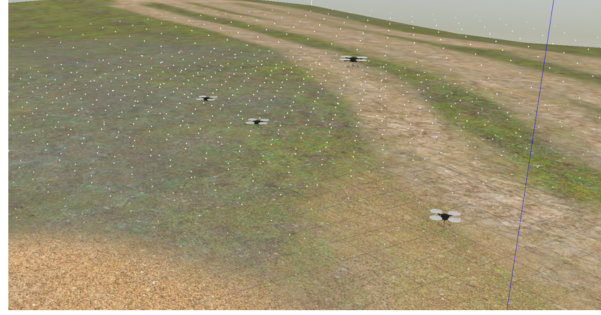


Figure 6.3. Visualization of the simulation environment with Gazebo.

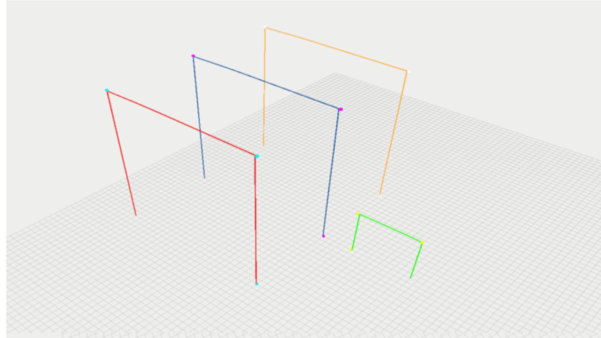


Figure 6.4. Visualization of the UAVs' pre-planned waypoints and paths with Rviz.

and communication subsystems while the second battery is used to power the flight control and propulsion subsystems of the UAV. The flight time of UAV with the battery fully charged is around 15 minutes and the operation time of the computing and communication unit with the battery fully charged is around 45 minutes.

6.5 Simulation Studies

In this section, we simulate NAC using the designed ROS- and Gazebo-based simulator with UAVs collaboratively completing the matrix multiplication task. We examine the impact of UAV mobility by comparing results from two scenarios: the *moving scenario* where UAVs move continuously during task execution and the *static scenario* where the UAVs remain stationary during task execution. The detailed simulation configurations and results are presented as follows.

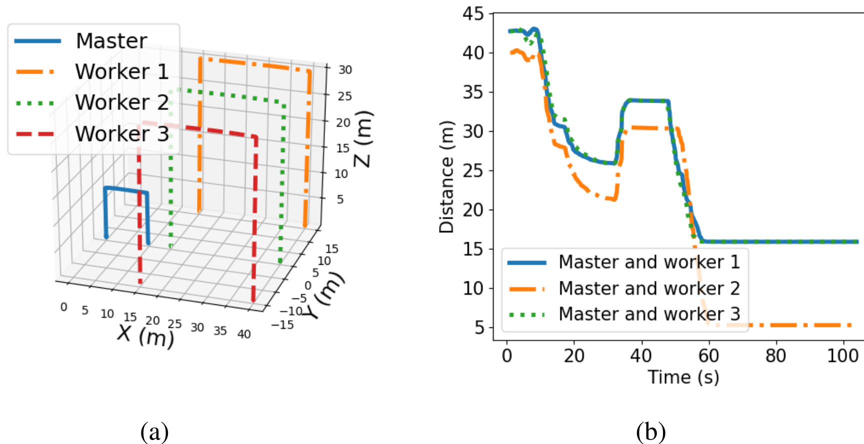


Figure 6.5. a) Trajectories of the four UAVs and b) distances between the master UAV and worker UAVs in the simulation.

6.5.1 Simulation Configurations

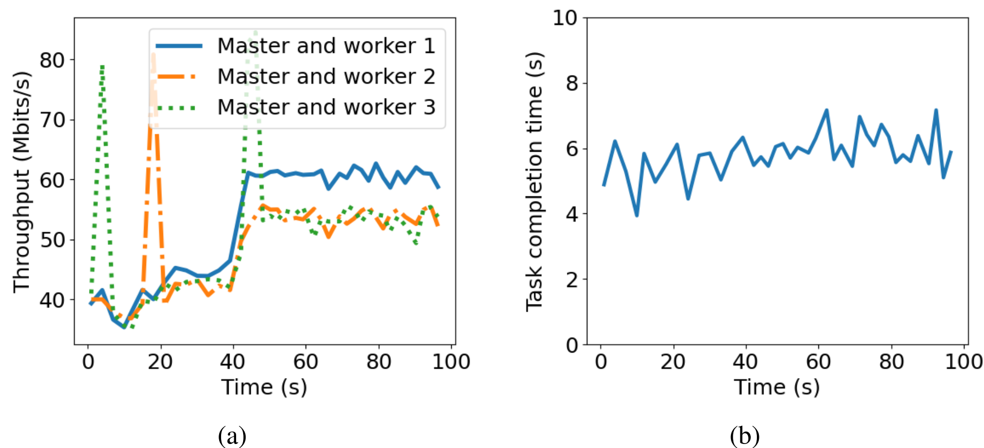
We simulate a NAC system with four UAVs. The computation task considered is matrix multiplication as described in Sec. 6.2. The matrices A and B are randomly generated and have a dimension of 100×10000 and 10000×1, respectively. This matrix multiplication task is conducted repeatedly for 40 iterations. The parameters of the position controller described in (6.1) are set to $k_x = 0.15, k_y = 0.15, k_z = 1$. The parameters of the wireless communication module as described in (6.2), (6.4) and (6.5) are set to $S_t = 1200MHz, f_c = 1200MHz, W = 1200MHz, N_0 = 1.1 \times 10^{-35}$ watts. In the moving scenario, UAVs are controlled to follow pre-planned waypoints (see Figs. 6.3-6.4 as an illustration). In the static scenario, UAVs are placed on the ground.

6.5.2 Simulation Results

The simulation results, including the throughput and computation time, for both the moving and static scenarios, are presented as follows.

Moving Scenario

The trajectories of the four UAVs in the moving scenario are shown in Fig. 6.5(a). Each UAV takes off from the ground, flies straightly at a constant altitude, and finally lands. The



(a) (b)

Figure 6.6. a) Throughput between the master and worker UAVs and b) completion time for a single iteration of matrix multiplication in the moving scenario.

master UAV flies a shorter distance than the worker UAVs to vary the distances between them. Fig. 6.5(b) shows the distances between the master and worker UAVs. As we can see, the distances between the master and workers decrease during the first 35s, remain stable for 15s, and then decrease again until landing at around 60s.

As shown in Fig. 6.6(a), the varying distances between the master and workers result in changes in the throughputs. The throughputs increase over time as the distances decrease. It is also noted that there are significant variations in the throughputs due to UAVs' mobility. Fig. 6.6(b) shows the completion time for a single iteration of matrix multiplication, which has an average value of around 5s.

Static Scenario

In the static scenario, the distances between the master and worker UAVs are set to 9.43m, 12.80m, and 17m. Fig. 6.7(a) depicts the throughputs between UAVs. Compared to Fig. 6.6(a), we can see that the throughputs between the static UAVs have smaller variances, resulting in more stable communication between the UAVs. The time for completing a single iteration of matrix multiplication in the static scenario is shown in Fig. 6.7(b).

To further understand the relationship between distance and communication performance, we vary the distance between two UAVs and measure the throughput. The results, which include

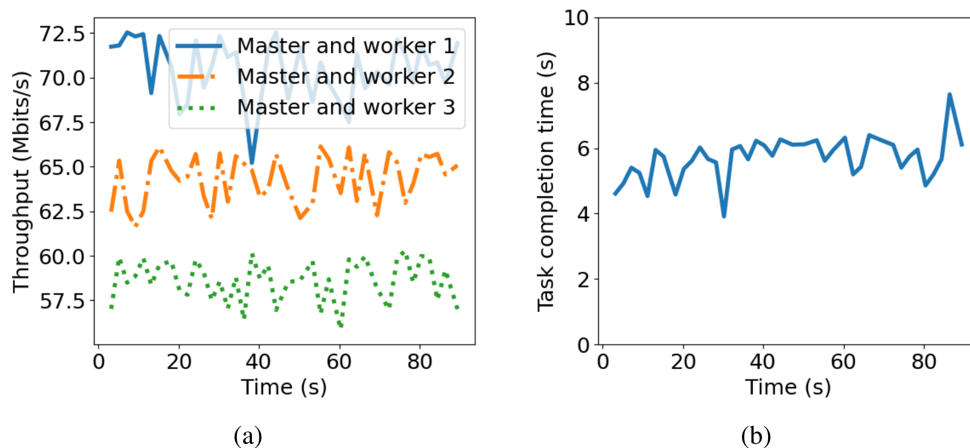


Figure 6.7. a) Throughput between the master and workers and b) completion time for a single iteration of matrix multiplication in the static scenario.

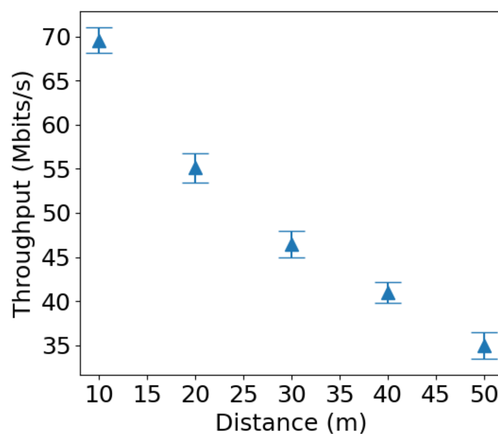


Figure 6.8. Throughput between two UAVs at various distances in simulations.

the mean values and standard deviations of 100 data samples, are shown in Fig. 6.8. As we can see, the throughput decreases quickly as the distance between the UAVs increases. This graph demonstrates the crucial impact of UAV mobility in NAC.

6.6 Real Flight Tests

In this section, we conduct real flight tests at San Diego State University (SDSU) sport field as shown in Fig. 6.9 to evaluate the NAC hardware testbed for performing linear regression. The experiments include both moving and static scenarios to assess the impact of UAV mobility on the system performance. The configurations and results of the experiments are described in

Figure 6.9. Flight test of NAC hardware testbed with three UAVs at the San Diego State University (SDSU) sport field.

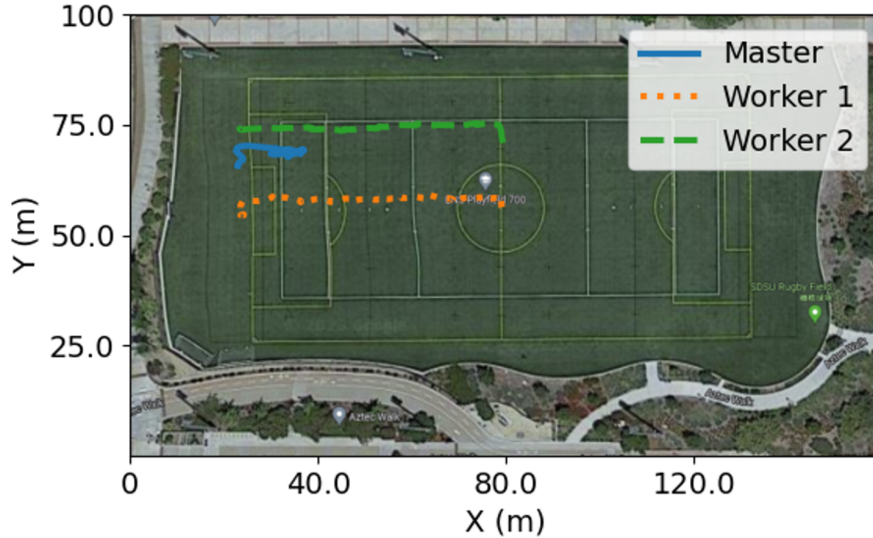


Figure 6.10. Trajectories of the three UAVs.

detail below.

6.6.1 Experiment Configurations

In the flight tests, each UAV in the moving scenario follows pre-planned waypoints with a flying speed of 0.67 m/s and a constant altitude of 3.04m. In the static scenario, all UAVs remain stationary on the ground. The computation task performed by the UAVs in both scenarios is to train a linear regressor for forest fire detection as described in Sec. 6.2. The throughput between two UAVs is measured using $Iperf3$ [235].

6.6.2 Experiment Results

Moving Scenario

We extract the positions of UAVs from the flight logs and calculate the distance between the master UAV and the two worker UAVs. The trajectories of the UAVs are displayed in Fig. 6.10. The two workers follow straight lines in parallel, while the master flies a short distance in between them and then hovers until the completion of the task.

The distances between master UAV and worker UAVs are shown in Fig. 6.11(a). The corresponding throughputs between UAVs are depicted in Fig. 6.11(b). It can be observed that within the first 40*s*, the throughput between master and worker 2 increases even though the distance between them remains relatively stable. This variation in the throughput may be attributed to the impact of UAV mobility and wireless communication interference. Moreover, the throughput between master and worker 2 is much larger than that between master and worker 1 during the first 40s, due to the shorter distance between them. After this point, both throughputs decrease drastically as the distances continuously increase and approach zero when the distances reach around 20m. The training terminates after 75*s* due to communication loss caused by increasing distances.

Fig. 6.12. presents the impact of distance on communication performance. As expected, the throughput decreases as the distance between the UAVs increases. The maximum communication distance between two UAVs is around 25m as indicated in the figure.

Static Scenario

To further evaluate the impact of mobility on the communication and computation performance, we keep all UAVs stationary on the ground and re-run the forest fire detection application. This allows us to compare and contrast the results with those obtained from the moving scenario.

In this scenario, we set the distances between the master and worker 1 and 2 to 9.8m and 8.9m, respectively. The throughputs between the UAVs are shown in Fig. 6.14(a). Compared to

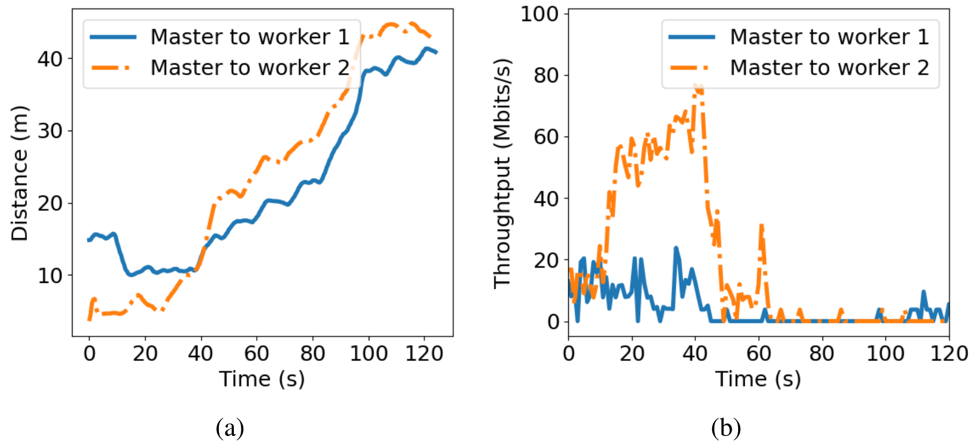


Figure 6.11. a) Distance and b) throughput between the master UAV and worker UAVs in the moving scenario.

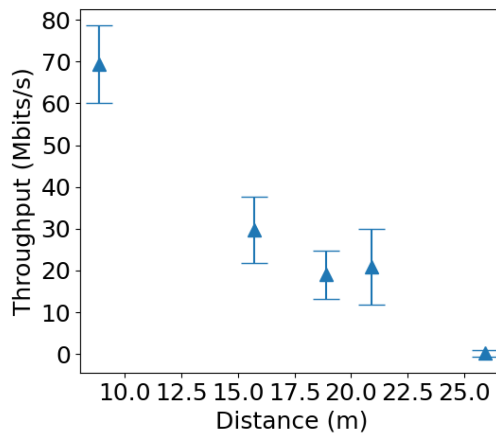


Figure 6.12. Throughput between two UAVs at various distances in flight tests.

the throughputs when UAVs are moving as shown in Fig. 6.11(b), we can see that, during the first 40s, the throughput between the master and worker 2 is lower in the static case due to the larger distance. However, the throughput variance is smaller when the UAVs are static.

We further analyze the computation performance by showing the training time for 10 iterations and the training cost in Fig. 6.14(b) and Fig. 6.14(c), respectively. The results indicate that the task completion time is generally smaller in the static case compared to the case when UAVs are moving due to shorter inter-vehicle distances as shown in Fig. 6.13(a). Moreover, the training completes 640 iterations within 200s when UAVs are static, while only 310 iterations are completed in the moving case within 75s. This is due to the fact that, in the moving scenario,

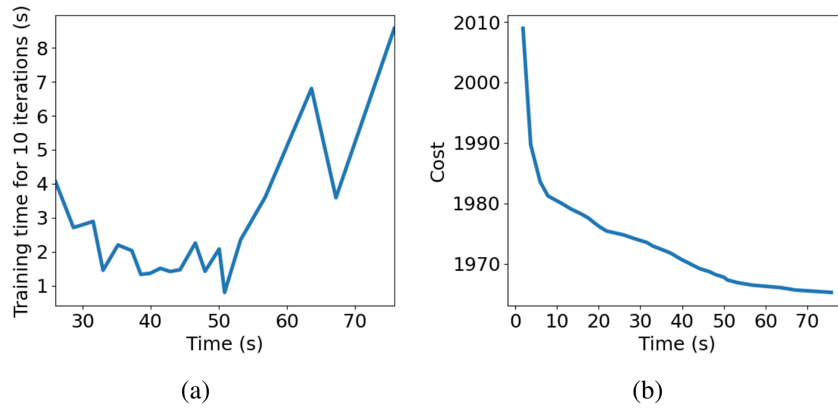


Figure 6.13. a) Time for completing 10 training iterations and b) the associated training cost in the moving scenario.

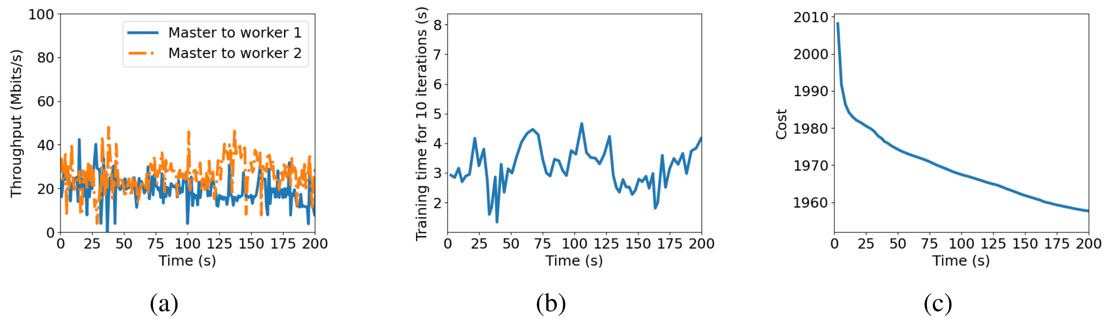


Figure 6.14. a) Throughput between the master UAV and worker UAVs in the static scenario; b) Time for completing 10 training iterations and c) the associated training cost in the static scenario.

the training terminates when workers move out of the master's communication range, which occurs at 75s. As more training iterations are performed, the training cost is also smaller in the static case as shown in Fig. 6.13(b) and Fig. 6.14(c).

6.7 Discussions and Conclusions

The simulation and flight tests reveal challenges in achieving high-performance NAC. In particular, UAV mobility results in significant communication variance, which can lead to The flight tests also demonstrate the limitations of standard Wi-Fi routers with omni-directional antennas. They have limited communication range and do not provide sufficient communication throughput for real-time applications. To address these limitations, our future work will focus on

improving the UAV-to-UAV communication range and throughput through the use of phased array antennas and mmWave 5G technology [222].

In conclusion, this chapter introduces a realistic simulator and hardware testbed for NAC research. The simulator, built using ROS and Gazebo, emulates networked UAVs with inter-vehicle resource sharing and distributed computing capabilities and can simulate various realistic environments. The hardware testbed consists of three quadrotors each with a Pixhawk control unit, Jetson TX2 computing unit, and telemetry radio and TP-Link Wi-Fi router for communication. The simulation and real flight tests on two computation applications showed that UAV mobility has a direct impact on the throughput, resulting in degraded and unstable communication performance as distance increases.

6.8 Acknowledgement

This chapter, in full, has been submitted for publication of the material as it may appear in: B. Wang, J. Xie, K. Ma, Y. Wan "UAV-based Networked Airborne Computing Simulator and Testbed Design and Implementation", *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Networked Airborne Computing is a new computing paradigm that has the potential to significantly improve the onboard computing capabilities of UAVs to support advanced UAVs applications as well as provide computing services to ground users. Due to unique features of UAVs such as high maneuverability and small load, enabling NAC is challenging because of its unique features such as high mobility, small load, and uncertain operating environment. This dissertation introduces innovative techniques to address these challenges, as a step towards enabling NAC.

Starting from designing a single UAV platform for NAC, Chapter 2 designs a new UAV-based airborne computing platform to address the onboard computing limitations of existing UAV platforms. This airborne computing platform was designed from two aspects: hardware and software. To design the hardware, we first investigated the desired features for the onboard computing hardware, and then conducted a comprehensive comparison study among state-of-the-art single-board computers to select a suitable one as the computing unit. A prototype was then designed and implemented, which not only contains the computing unit, but also hardware for UAS mobility, communications and control. To design the software, we investigated two representative virtualization techniques, VM using KVM and container using Docker, and evaluated their performances from various aspects. Through comprehensive experimental studies,

we find that Docker outperforms KVM in most performance aspects, including computing, networking, isolation of most hardware resources, power consumption, and resource usage. Docker also successfully virtualizes all CPU cores and GPU in Jetson TX2. On the other hand, KVM is more secure.

NAC consists of multiple UAV platforms, which can further enhance computing capabilities by computing resources sharing and distributed computing. Chapter 3 proposes a novel CDC framework, namely, BPCC, for NAC. The key idea of BPCC is to optimally exploit partial coded results calculated by all distributed computing nodes in NAC. Under this BPCC framework, we then investigated a classical CDC problem, matrix-vector multiplication, and formulated an optimization problem for BPCC to minimize the expected task completion time, by configuring the computation load. The BPCC was proved to provide an asymptotically optimal solution and outperform a state-of-the-art CDC scheme for heterogeneous clusters, namely, *heterogeneous coded matrix multiplication* (HCMM). Theoretical analysis reveals the impact of BPCC's key parameter, i.e., number of batches, on its performance, the results of which infer the worst and best performance that BPCC can achieve. To evaluate the performance of the proposed BPCC scheme and better understand the impacts of its parameters, we conducted extensive simulation studies and real experiments on the Amazon EC2 computing clusters. The simulation and experimental results verify theoretical results and also demonstrate that the proposed BPCC scheme outperforms all benchmark schemes in computing systems with uncertain stragglers, in terms of the task completion time and robustness to stragglers.

Chapter 4 further extends BPCC to consider mobility of UAVs in NAC and proposes D-BPCC (Dynamic-BPCC) to enable efficient, robust, and adaptable cooperative airborne computing in the dynamic, heterogeneous, and uncertain airspace. To optimize system performance, DRL based online decision-making strategies are then designed for two typical NAC formation scenarios, which do not rely on perfect communication, computation or UAV mobility models. Simulation results show that our methods are more resilient to uncertain system disturbances than existing solutions, including the UU, LBU, HCMM, and C3P schemes, and are adaptive to

network topology and resource changes. Moreover, the effectiveness of our method in solving scenarios where NAC is formed by controllable UAVs demonstrates the benefits of UAV mobility control to robust computing.

Chapter 5 explores applications of NAC in MARL. In particular, we present DARL1N, a scalable MARL algorithm that can be trained over a distribute computing architecture. DARL1N reduces the representation complexity of the value and policy functions of each agent in a MARL problem by disregarding the influence of other agents that are not within one hop of a proximity graph. This model enables highly efficient distributed training, in which a compute node only needs data from an agent it is training and its potential one-hop neighbors. We conducted comprehensive experiments using five MARL environments and compared DARL1N with four state-of-the-art MARL algorithms. DARL1N generates equally good or even better policies in almost all scenarios with significantly higher training efficiency than benchmark methods, especially in large-scale problem settings. To improve the resilience of DARL1N to stragglers common in distributed computing systems, we developed coding schemes that assign each agent to multiple learners. The properties of MDS, Random Sparse, Repetition, LDPC, and LDGM codes were evaluated. Our results show that the MDS and Random Sparse codes offer high resilience to large numbers of stragglers. However, both schemes involve many redundant computations and have low training efficiency. The Repetition, LDGM, and LDPC coding schemes achieve better performance in both training efficiency and policy quality but can handle fewer stragglers in the system.

Chapter 6 introduces a realistic simulator and hardware testbed for NAC research. The simulator, built using ROS and Gazebo, emulates networked UAVs with inter-vehicle resource sharing and distributed computing capabilities and can simulate in various realistic environments. The hardware testbed consists of three quadrotors each with a Pixhawk control unit, Jetson TX2 computing unit, and telemetry radio and TP-Link Wi-Fi router for communication. The simulation and real flight tests on two computation applications showed that UAV mobility has a direct impact on the throughput, resulting in degraded and unstable communication performance

as distance increases.

7.2 Future Work

There are several suggestions for future work. In Chapter 2, UAV platform adopts directional antenna to achieve long range and broadband communication, which, however, requires that two antennas's directions are physically adjusted to face to each other. To support more flexible communication, phased array antennas can be used to adjust directions without physically moving antennas. Moreover, KVM and Docker based live migration that allows migrating a running container or virtual machine from one UAV to another UAV without interruption can be investigated to improve the robustness and flexibility of single UAV platform.

For BPCC and D-BPCC methods proposed in Chapter 3 and 4, respectively, energy consumption or storage can be considered for optimization in addition to task completion time and UAV flight time.

In Chapter 5, both DARL1N and Coded DARL1N adopt centralized training architecture in which there is a central controller that collects and sends training parameters from and to workers. This centralized training architecture relies on central controller for successful training. In the future, a fully decentralized training architecture can be explored to improve the system robustness and flexibility.

For hardware testbed design in Chapter 6, the UAV-to-UAV communication range and throughput can be improved through the use of phased array antennas and mmWave 5G technology [222].

Appendix A

Proofs of Chapter 3

A.1 Proof of Lemma 1

To derive the infimum and supremum of λ_i, we first prove that they are attained at $p_i \to \infty$ and $p_i = 1$, respectively. Define the following auxiliary function

$$h_i(x,y) = (1+\frac{\mu_i x}{y})e^{-\mu_i(\frac{x}{y}-\alpha_i)}, \tag{A.1}$$

where $x > 0$ and $y \in [0,1]$. Note that $h_i(x,y)$ is a monotonically increasing function with respect to y, as $\frac{\partial h_i(x,y)}{\partial y} = \frac{\mu_i x}{y^3}e^{-\mu_i(\frac{x}{y}-\alpha_i)} > 0$. Also define $P(z) = \{y_0, y_1, y_2, \ldots, y_z\}$ as a *partition* of the range of y, i.e., $[0,1]$, with $y_k = \frac{k}{z}$, $z \in \mathbb{Z}^+$, $k \in \{0\} \cup [z]$, and let

$$M_k(x) = \sup_y \{h_i(x,y) |\, y_{k-1} \le y \le y_k\} = h_i(x,y_k) = (1+\frac{\mu_i xz}{k})e^{-\frac{\mu_i xz}{k}+\mu_i\alpha_i} \tag{A.2}$$

for each $k \in [z]$. We then let $\Delta y_k = y_k - y_{k-1} = \frac{1}{z}$, $k \in [z]$, and define

$$U(z,x) = \sum_{k=1}^{z} M_k(x)\Delta y_k = \frac{1}{z}\sum_{k=1}^{z}(1+\frac{\mu_i xz}{k})e^{-\frac{\mu_i xz}{k}+\mu_i\alpha_i}$$

According to **Theorem 6.4** in [236], if there exists another partition $P(z')$ that satisfies $P(z') \supset P(z)$, then

$$U\left(z',x\right) < U(z,x) \tag{A.3}$$

We can then derive that

$$U(\infty,x) < U(z,x) \leq U(1,x), \tag{A.4}$$

as $P(z) \supseteq P(1)$ and $P(z) \subset P(\infty)$, $\forall z \in \mathbb{Z}^+$. The right equality holds when $z = 1$. Furthermore, as $\frac{\partial U(z,x)}{\partial x} = \frac{1}{z}\sum_{k=1}^{z} -\frac{u_i^2 zx}{k^2} e^{-\frac{u_i zx}{k}+\mu_i\alpha_i} < 0$, $U(z,x)$ is a monotonically decreasing function with respect to x.

Now let $x = \lambda_i$ and $z = p_i$. We then have

$$U(p_i,\lambda_i) = \frac{1}{p_i}\sum_{k=1}^{p_i}(1+\frac{\mu_i\lambda_i p_i}{k})e^{-\frac{\mu_i\lambda_i p_i}{k}+\mu_i\alpha_i} = 1, \tag{A.5}$$

according to Eq. (7). Using proof of contradiction, we can then derive that the infimum and supremum of λ_i are attained when $p_i \to \infty$ and $p_i = 1$, respectively. Specifically, suppose $\sup\lambda_i = \bar{\lambda}$ is attained at $p_i = \bar{p} > 1$ and λ^* is attained at $p_i = 1$, we then have $\bar{\lambda} \geq \lambda^*$ and $U(1,\bar{\lambda}) \leq U(1,\lambda^*)$ as $U(z,x)$ is a monotonically decreasing function with respect to x. Since $U(\bar{p},\bar{\lambda}) = U(1,\lambda^*)$ according to Eq. (A.5), we have $U(1,\bar{\lambda}) \leq U(\bar{p},\bar{\lambda})$, which contradicts with $U(1,\bar{\lambda}) > U(\bar{p},\bar{\lambda})$ according to Eq. (A.4). Therefore, $\sup\lambda_i$ must be attained at $p_i = 1$. Similarly, we can prove that the infimum of λ_i is attained when $p_i \to \infty$.

Next, we find the specific formulas for the infimum and supremum of λ_i. In particular, the infimum of λ_i is attained when $p_i \to \infty$, which can be found by solving Eq. (A.5). Specifically,

$$\lim_{p_i\to\infty}\frac{1}{p_i}\sum_{k=1}^{p_i}(1+\frac{\mu_i\lambda_i p_i}{k})e^{-\frac{\mu_i\lambda_i p_i}{k}+\mu_i\alpha_i} = 1 \tag{A.6}$$

which is equivalent to solving

$$\int_0^1 (1+\frac{\mu_i\lambda_i}{x})e^{\frac{-\mu_i\lambda_i}{x}}dx = e^{-\mu_i\alpha_i} \tag{A.7}$$

Define variable $v = \frac{\mu_i\lambda_i}{x}$, the term in the left side of the above equation can be simplified as

$$
\begin{aligned}
&\int_0^1 \left(1+\frac{\mu_i\lambda_i}{x}\right) e^{-\frac{\mu_i\lambda_i}{x}} dx \\
=&-\mu_i\lambda_i \int_{\mu_i\lambda_i}^{\infty} (1+v)e^{-v} d\left(\frac{1}{v}\right) \\
=&-\mu_i\lambda_i \left[(\frac{1}{v}+1)e^{-v}|_{\mu_i\lambda_i}^{\infty} - \int_{\mu_i\lambda_i}^{\infty} \frac{1}{v} d((1+v)e^{-v}) \right] \\
=&(1+\mu_i\lambda_i)e^{-\mu_i\lambda_i} - \mu_i\lambda_i \int_{\mu_i\lambda_i}^{\infty} e^{-v} dv \\
=&e^{-\mu_i\lambda_i}
\end{aligned}
\tag{A.8}
$$

Combining Eq. (A.7) and Eq. (A.8), we can get $\lambda_i = \alpha_i$, when $p_i \to \infty$.

Similarly, we can obtain the supremum of λ_i by solving Eq. (A.5) and setting $p_i = 1$. Specifically, we aim to solve the following equation

$$
(1+\mu_i\lambda_i)e^{-\mu_i\lambda_i+\mu_i\alpha_i} = 1 \tag{A.9}
$$

According to [130], the solution to $a^x = x+b$ is $x = \frac{-b-W(-a^{-b}\ln a)}{\ln a}$, where $W(\cdot)$ is the Lambert W function. We can then get the following solution to the above equation

$$
\lambda_i = \frac{W(-e^{-\alpha_i\mu_i-1})+1}{-\mu_i},
$$

by letting $x = \mu_i\lambda_i - \alpha_i\mu_i$, $a = e$ and $b = \alpha_i\mu_i + 1$.

A.2 Proof of Lemma 2

To prove $\tau^* - o(1) < t^* \leq \tau^* + o(1)$ in Lemma 2, we will apply the McDiarmid's inequalities [237] described as follows. For a set of independently distributed random variables,

$x_1, x_2, \ldots, x_N \in \mathscr{X}$, if a function $f: \mathscr{X}^N \rightarrow \mathbb{R}$ satisfies the Lipschitz condition:

$$\left|f\left(x_1, \ldots, x_i, \ldots, x_N\right) - f\left(x_1, \ldots, x_i', \ldots, x_N\right)\right| \leq c_i,$$

for all $x_1, \ldots, x_N, x_i' \in \mathscr{X}$, then, for any $\sigma > 0$,

$$\Pr\left[\mathbb{E}[f(X)] - f(X) \geq \sigma\right] \leq e^{-\frac{2\sigma^2}{\Sigma_{i=1}^N c_i^2}} \quad \text{(A.10)}$$

$$\Pr\left[f(X) - \mathbb{E}[f(X)] \geq \sigma\right] \leq e^{-\frac{2\sigma^2}{\Sigma_{i=1}^N c_i^2}} \quad \text{(A.11)}$$

where $X = (x_1, x_2, \ldots, x_N) \in \mathscr{X}^N$. To apply the above McDiarmid's inequalities in our problem, we define $x_i(t) = s_i(t)b_i, \forall i \in [N]$, and further define

$$X(t) = \sum_{i=1}^{N} x_i(t) = \sum_{i=1}^{N} s_i(t) b_i = S(t).$$

Clearly, under such definitions, we have $c_i(t) = \ell_i(t)$. To facilitate further discussions, we let $\delta = \Theta\left(\frac{\log N}{\sqrt{N}}\right) = o(1)$, $\varepsilon = \delta^2$. We also summarize the asymptotic scales for the parameters: $r = \Theta(N)$, $\lambda_i = \Theta(1)$, $\beta = \Theta(N)$, $\tau^* = \Theta(1)$, $\ell_i^*(\tau^*) = \Theta(1)$.

Now, let's prove the first inequality in Lemma 2: $\tau^* - o(1) < t^*$. Define $t = \tau^* - \delta$. According to Eq. (12), we can derive

$$\beta t = \beta\tau^* - \beta\delta = r - \beta\delta.$$

Applying the second McDiarmid's inequality in Eq. (A.11), we can then derive

$$\begin{aligned}
&\Pr\left[S^*(t) \geq r+\varepsilon\right] \\
=&\Pr\left[S^*(t) \geq \mathbb{E}[S^*(t)] - \mathbb{E}[S^*(t)] + r + \varepsilon\right] \\
=&\Pr\left[S^*(t) \geq \mathbb{E}[S^*(t)] - \beta t + r + \varepsilon\right] \\
=&\Pr\left[S^*(t) - \mathbb{E}[S^*(t)] \geq \beta\delta + \varepsilon\right] \\
\leq& e^{-\frac{2(\beta\delta+\varepsilon)^2}{\sum_{i=1}^{N}(\ell_i^*(t))^2}}
\end{aligned} \tag{A.12}$$

Using the asymptotic scales of parameters in the right hand side of Ineq. (A.12), we have

$$\Pr\left[S^*(t) \geq r+\varepsilon\right] \leq \Theta(e^{-\log^2 N}). \tag{A.13}$$

Consequently, we have

$$\Pr\left[S^*(t) < r+\varepsilon\right] > 1 - \Theta(e^{-\log^2 N}) = \Theta(1). \tag{A.14}$$

Ineq. (A.14) shows that, if $t^* \leq \tau^* - \delta$, then the probability $\Pr\left[S^*(t^*) < r+\varepsilon\right]$ is not $o(\frac{1}{N})$, which does not satisfy the constraint in $\mathscr{P}_{\text{alt}}^{(2)}$. Therefore, $t^* > \tau^* - o(1)$.

Next, we prove the second inequality in Lemma 2: $t^* \leq \tau^* + o(1)$. Define $t' = \tau^* + \delta$. According to Eq. (12), we can derive

$$\beta t' = \beta\tau^* + \beta\delta = r + \beta\delta.$$

Applying the first McDiarmid's inequality in Eq. (A.10), we can then derive

$$\begin{aligned}
&\Pr\left[S^*(t') \le r-\varepsilon\right] \\
=&\Pr\left[S^*(t') \le \mathbb{E}[S^*(t')] - \mathbb{E}[S^*(t')] + r - \varepsilon\right] \\
=&\Pr\left[S^*(t') \le \mathbb{E}[S^*(t')] - \beta t' + r - \varepsilon\right] \\
=&\Pr\left[\mathbb{E}[S^*(t')] - S^*(t') \ge \beta\delta + \varepsilon\right] \\
\le& e^{-\frac{2(\beta\delta+\varepsilon)^2}{\sum_{i=1}^{N}(\ell_i^*(t))^2}}
\end{aligned} \tag{A.15}$$

Using the asymptotic scales of parameters in the right hand side of Ineq. (A.15), we have

$$\Pr\left[S^*(t') \le r-\varepsilon\right] \le \Theta(e^{-\log^2 N}).$$

Ineq. (A.15) shows that $t' = \tau^* + \delta$ can satisfy the constraint in $\mathscr{P}_{\text{alt}}^{(2)}$. Since t^* is the minimal time that satisfies the constraint, $t^* \le t' = \tau^* + \delta$. Therefore, $t^* \le \tau^* + o(1)$.

Proof of Theorem 3

The asymptotic optimality of BPCC shown in Eq. (16) can be proved by showing that

$$t^* - o(1) \le \mathbb{E}\left[T_{\text{OPT}}\right] \le \mathbb{E}\left[T_{\text{BPCC}}\right] \le t^* + o(1)$$

Since $\mathbb{E}\left[T_{\text{OPT}}\right] \le \mathbb{E}\left[T_{\text{BPCC}}\right]$ is straightforward because $\mathbb{E}[T_{\text{OPT}}]$ is the optimal value of P'_{main}, we use two steps, inspired by [25], to prove the other two inequalities.

Step 1: To prove $t^* - o(1) \le \mathbb{E}\left[T_{\text{OPT}}\right]$.

Let $\ell_{\text{OPT}} = (\ell_{\text{OPT},1}, \cdots, \ell_{\text{OPT},N})$ be the optimal load allocation obtained by solving $\mathscr{P}'_{\text{main}}$ and let $S_{\text{OPT}}(t)$ be the amount of results received by the master node by time t under load

allocation ℓ_{OPT}. The inequality above can be proved by showing the following inequalities:

$$t^* - \delta_2 - \delta_1 \overset{(b)}{\leq} \tau^*_{\text{OPT}} - \delta_1 \overset{(a)}{\leq} \mathbb{E}\left[T_{\text{OPT}}\right]$$

where τ^*_{OPT} is the solution to $\mathbb{E}\left[S_{\text{OPT}}(t)\right] = r$, and δ_1 and δ_2 are both $\Theta\left(\frac{\log N}{\sqrt{N}}\right) = o(1)$. To prove Ineq. (a), we first define an auxiliary function $g_i(t)$ for each node i as

$$g_i(t) = 1 - \frac{1}{p_i}\sum_{k=1}^{p_i} e^{-\mu_i\left(\frac{tp_i}{k\ell_{OPT,i}} - \alpha_i\right)}.$$

According to Eq. (5), we have

$$\mathbb{E}\left[S_{\text{OPT}}(t)\right] = \sum_{i=1}^{N} \ell_{\text{OPT},i}\, g_i(t)$$

and

$$\begin{aligned}
& r - \mathbb{E}[S_{\text{OPT}}\left(\tau^*_{\text{OPT}} - \delta_1\right)] \\
= & \ \mathbb{E}\left[S_{\text{OPT}}(\tau^*_{\text{OPT}})\right] - \mathbb{E}[S_{\text{OPT}}\left(\tau^*_{\text{OPT}} - \delta_1\right)] \\
= & \sum_{i=1}^{N} \ell_{\text{OPT},i}\left[g_i(\tau^*_{\text{OPT}}) - g_i(\tau^*_{\text{OPT}} - \delta_1)\right] \\
= & \sum_{i=1}^{N} \ell_{\text{OPT},i}\left(\frac{dg_i(\tau^*_{\text{OPT}})}{d\tau^*_{\text{OPT}}}\delta_1 + \mathscr{O}\left(\delta_1^2\right)\right)
\end{aligned}$$

According to our previous discussions, $r = \Theta(N)$, so $\ell_{OPT,i} = \Theta(1)$, $\forall i \in [N]$. Therefore, $g_i(t)$ does not change with N, i.e., $g_i(t) = \Theta(1)$. We then have

$$\begin{aligned}
r - \mathbb{E}\left[S_{\text{OPT}}\left(\tau^*_{\text{OPT}} - \delta_1\right)\right] &= \Theta\left(N\delta_1\right) + \mathscr{O}\left(N\delta_1^2\right) \\
&= \Theta\left(N\delta_1\right)
\end{aligned}$$

By using the McDiarmid's inequality in Eq. (A.11), we have

$$
\begin{aligned}
&\Pr\left[S_{\mathrm{OPT}}\left(\tau^*_{\mathrm{OPT}}-\delta_1\right)\geq r\right] \\
=&\Pr\{S_{\mathrm{OPT}}\left(\tau^*_{\mathrm{OPT}}-\delta_1\right)-\mathbb{E}\left[S_{\mathrm{OPT}}\left(\tau^*_{\mathrm{OPT}}-\delta_1\right)\right]\geq \\
&r-\mathbb{E}\left[S_{\mathrm{OPT}}\left(\tau^*_{\mathrm{OPT}}-\delta_1\right)\right\} \\
\leq& e^{-\frac{2\left(\mathbb{E}\left[S_{\mathrm{OPT}}\left(\tau^*_{\mathrm{OPT}}-\delta_1\right)\right]-r\right)^2}{\Sigma_{i=1}^{N}\ell^2_{\mathrm{OPT},i}}} \\
=& e^{-\Theta\left(N\delta_1^2\right)}=o\left(\frac{1}{N}\right),
\end{aligned}
$$

which implies that $\mathbb{E}[T_{\mathrm{OPT}}]\geq\tau^*_{\mathrm{OPT}}-\delta_1$.

Next, we proceed to prove Ineq. (b). Since $\mathbb{E}[S^*(t)]$ is the optimal value of $P^{(1)}_{\mathrm{alt}}$, we have $\mathbb{E}[S^*(t)]\geq\mathbb{E}[S_{\mathrm{OPT}}(t)]$. Moreover, as $\mathbb{E}[S^*(\tau^*)]=r$ according to Eq. (10), $\mathbb{E}[S_{\mathrm{OPT}}(\tau^*_{\mathrm{OPT}})]=r$, and both $\mathbb{E}[S^*(t)]$ and $\mathbb{E}[S_{\mathrm{OPT}}(t)]$ increase monotonically with t, we can derive

$$\tau^*_{\mathrm{OPT}}\geq\tau^*$$

According to Lemma 2,

$$\tau^*\geq t^*-\delta_2$$

Therefore,

$$\tau^*_{\mathrm{OPT}}-\delta_1\geq t^*-\delta_1-\delta_2$$

We have now proved $t^*-o(1)\leq\mathbb{E}\left[T_{\mathrm{OPT}}\right]$.

Step 2: To prove $\mathbb{E}\left[T_{\mathrm{BPCC}}\right]\leq t^*+o(1)$.

Let $T_{\max}$ be a random variable that denotes the time required for all worker nodes to complete their tasks assigned using BPCC. Let $\mathscr{E}_1=\{T_{\max}>\Theta(N)\}$ and $\mathscr{E}_2=\{T_{\mathrm{BPCC}}>t^*\}$ be two events. $\mathbb{E}[T_{\mathrm{BPCC}}]$ can then be computed by

$$\begin{aligned}\mathbb{E}\left[T_{\text{BPCC}}\right] =& \mathbb{E}\left[T_{\text{BPCC}}|\mathscr{E}_1\right]\Pr\left[\mathscr{E}_1\right] \\ &+ \mathbb{E}\left[T_{\text{BPCC}}|\mathscr{E}_1^c \cap \mathscr{E}_2\right]\Pr\left[\mathscr{E}_1^c \cap \mathscr{E}_2\right] \\ &+ \mathbb{E}\left[T_{\text{BPCC}}|\mathscr{E}_1^c \cap \mathscr{E}_2^c\right]\Pr\left[\mathscr{E}_1^c \cap \mathscr{E}_2^c\right] \qquad \text{(A.16)}\end{aligned}$$

The first term in the right hand side of Eq. (A.16) can be written as

$$\begin{aligned}&\mathbb{E}\left[T_{\text{BPCC}}|\mathscr{E}_1\right]\Pr\left[\mathscr{E}_1\right] \\ &= \mathbb{E}\left[T_{\text{BPCC}}|T_{\max} > \Theta(N)\right] \times \Pr\left[T_{\max} > \Theta(N)\right] \\ &\leq \mathbb{E}\left[T_{\max}|T_{\max} > \Theta(N)\right] \times \Pr\left[T_{\max} > \Theta(N)\right] \\ &= \int_{\Theta(N)}^{\infty} t f_{\max}(t) dt, \qquad \text{(A.17)}\end{aligned}$$

where $f_{\max}(t)$ is the probability density function (PDF) of $T_{\max}$. A stochastic upper bound of $T_{\max}$ can be found by using N worker nodes that all take the smallest straggling parameter $\min\{\mu_i\}$ and the largest shift parameter $\max\{\alpha_i\}$. Using the PDF of the maximum of N i.i.d. exponential random variables, we then have

$$\begin{aligned}&\mathbb{E}\left[T_{\text{BPCC}}|\mathscr{E}_1\right]\Pr\left[\mathscr{E}_1\right] \\ &\leq \int_{\Theta(N)}^{\infty} t f_{\max}(t) dt \\ &\leq \int_{\Theta(N)}^{\infty} tNk_1 e^{-k_1 t}\left(1 - e^{-k_1 t}\right)^{N-1} dt \\ &\leq \int_{\Theta(N)}^{\infty} Nk_1 t e^{-k_1 t} dt \\ &= -N(t + \frac{1}{k_1})e^{-k_1 t}\Big|_{t=\Theta(N)}^{\infty} \\ &= o(1) \qquad \text{(A.18)}\end{aligned}$$

where k_1 is a constant, i.e., $k_1 = \Theta(1)$.

The second term in the right hand side of Eq. (A.16) can be written as

$$\begin{aligned}
&\mathbb{E}[T_{\text{BPCC}}|\mathscr{E}_1^c \cap \mathscr{E}_2] \Pr[\mathscr{E}_1^c \cap \mathscr{E}_2] \\
&= \mathbb{E}[T_{\text{BPCC}}|T_{\max} \leq \Theta(N), T_{\text{BPCC}} > t^*] \\
&\quad \times \Pr[T_{\max} \leq \Theta(N), T_{\text{BPCC}} > t^*] \\
&\leq \mathbb{E}[T_{\max}|T_{\max} \leq \Theta(N), T_{\text{BPCC}} > t^*] \\
&\quad \times \Pr[T_{\text{BPCC}} > t^*]
\end{aligned} \tag{A.19}$$

where $\mathbb{E}[T_{\max}|T_{\max} \leq \Theta(N), T_{\text{BPCC}} > t^*]$ can be computed by

$$\begin{aligned}
&\mathbb{E}[T_{\max}|T_{\max} \leq \Theta(N), T_{\text{BPCC}} > t^*] \\
&= \frac{1}{\Pr[T_{\max} \leq \Theta(N), T_{\text{BPCC}} > t^*]} \\
&\quad \times \int_{t_1=0}^{\Theta(N)} \int_{t_2=t^*}^{\infty} t_1 d \Pr[T_{\max} \leq t_1, T_{\text{BPCC}} \leq t_2] \\
&\leq \frac{\Theta(N)}{\Pr[T_{\max} \leq \Theta(N), T_{\text{BPCC}} > t^*]} \\
&\quad \times \int_{t_1=0}^{\Theta(N)} \int_{t_2=t^*}^{\infty} d \Pr[T_{\max} \leq t_1, T_{\text{BPCC}} \leq t_2] \\
&= \Theta(N)
\end{aligned}$$

Since the master node receives at least r rows of inner product results by time T_{BPCC}, we have $S^*(T_{\text{BPCC}}) \geq r$. Next, since $S^*(t)$ is a monotonically increasing function with respect to time t, we can derive that, if $S^*(t^*) < r$, then $T_{\text{BPCC}} > t^*$, which leads to

$$\Pr[T_{\text{BPCC}} > t^*] \leq \Pr[S^*(t^*) < r] = o\left(\frac{1}{N}\right)$$

Therefore, Eq. (A.19) can be written as

$$\begin{aligned}\mathbb{E}\left[T_{\text{BPCC}}|\mathscr{E}_1^c \cap \mathscr{E}_2\right]\Pr\left[\mathscr{E}_1^c \cap \mathscr{E}_2\right] &\leq \Theta(N)\cdot o\left(\frac{1}{N}\right) \\ &= o(1) \end{aligned} \quad \text{(A.20)}$$

The third term in the right hand side of Eq. (A.16) can be written as:

$$\begin{aligned}
&\mathbb{E}[T_{\text{BPCC}}|\mathscr{E}_1^c \cap \mathscr{E}_2^c]\Pr\left[\mathscr{E}_1^c \cap \mathscr{E}_2^c\right] \\
&= \mathbb{E}\left[T_{\text{BPCC}}|T_{\max} \leq \Theta(N), T_{\text{BPCC}} \leq t^*\right] \\
&\quad \times \Pr\left[T_{\max} \leq \Theta(N), T_{\text{BPCC}} \leq t^*\right] \\
&\leq \mathbb{E}\left[T_{\text{BPCC}}|T_{\max} \leq \Theta(N), T_{\text{BPCC}} \leq t^*\right] \\
&= \frac{1}{\Pr\left[T_{\max} \leq \Theta(N), T_{\text{BPCC}} \leq t^*\right]} \\
&\quad \times \int_{t_1=0}^{\Theta(N)}\int_{t_2=0}^{t^*} t_2 d\Pr\left[T_{\max} \leq t_1, T_{\text{BPCC}} \leq t_2\right] \\
&\leq \frac{t^*}{\Pr\left[T_{\max} \leq \Theta(N), T_{\text{BPCC}} \leq t^*\right]} \\
&\quad \times \int_{t_1=0}^{\Theta(N)}\int_{t_2=0}^{t^*} d\Pr\left[T_{\max} \leq t_1, T_{\text{BPCC}} \leq t_2\right] \\
&= t^*
\end{aligned} \quad \text{(A.21)}$$

Combining Eq. (A.16), Eq. (A.18), Eq. (A.20), and Eq. (A.21), we then have $\mathbb{E}\left[T_{\text{BPCC}}\right] \leq t^* + o(1)$.

A.3 Proof of Theorem 4

According to Lemma 2 and Theorem 3, we can derive

$$\tau^* - o(1) \leq \mathbb{E}\left[T_{\text{BPCC}}\right] \leq \tau^* + o(1)$$

Therefore, $\lim_{N\to\infty}\mathbb{E}\left[T_{\text{BPCC}}\right]=\tau^*$.

Proof of Theorem 5

Before showing the proof of Theorem 5, we first present the following lemma, which will be used to prove this theorem.

Lemma 10. *Suppose $g(x)$ is a non-decreasing concave function and $g(x)\geq 0$, then*

$$\frac{1}{p}\sum_{k=1}^{p} g\left(\frac{k}{p}\right) \geq \frac{1}{p+1}\sum_{k=1}^{p+1} g\left(\frac{k}{p+1}\right), \tag{A.22}$$

for any $p\in\mathbb{Z}^+$.

Proof. Let $y_k=\frac{k}{p}$ and $z_k=\frac{k}{p+1}$, where $k\in[p]$. We then have

$$y_k=\left(1-\frac{k}{p}\right)z_k+\frac{k}{p}z_{k+1}. \tag{A.23}$$

As $g(x)$ is a concave function, we have

$$\begin{aligned} g\left(y_k\right) &= g\left[\left(1-\frac{k}{p}\right)z_k+\frac{k}{p}z_{k+1}\right] \\ &\geq \left(1-\frac{k}{p}\right)g\left(z_k\right)+\frac{k}{p}g\left(z_{k+1}\right) \end{aligned} \tag{A.24}$$

Moreover, as $g(x)$ is also a non-decreasing function, which implies $g(z_k)\leq g(z_{k+1})$, we then have

$$
\begin{aligned}
g(y_k) &\geq \left(1-\frac{k}{p}\right) g(z_k) + \frac{k}{p} g(z_{k+1}) \\
&= g(z_k) + \frac{k}{p}\left(g(z_{k+1}) - g(z_k)\right) \\
&\geq g(z_k) + \frac{k}{p+1}\left(g(z_{k+1}) - g(z_k)\right) \\
&= \left(1-\frac{k}{p+1}\right) g(z_k) + \frac{k}{p+1} g(z_{k+1})
\end{aligned} \tag{A.25}
$$

By summing over all possible values of k for both sides of the above inequality, we get

$$
\begin{aligned}
\sum_{k=1}^{p} g(y_k) \geq& \frac{1}{p+1} \sum_{k=1}^{p} (p+1-k) g(z_k) \\
&+ \frac{1}{p+1} \sum_{k=1}^{p} k g(z_{k+1}) \\
=& \frac{1}{p+1} [p \sum_{k=1}^{p} g(z_k) + \sum_{k=1}^{p} (1-k) g(z_k) \\
&+ p g(z_{p+1}) + \sum_{k=2}^{p} (k-1) g(z_k)] \\
=& \frac{p}{p+1} \sum_{k=1}^{p+1} g(z_k)
\end{aligned} \tag{A.26}
$$

which leads to $\frac{1}{p}\sum_{k=1}^{p} g\left(\frac{k}{p}\right) \geq \frac{1}{p+1}\sum_{k=1}^{p+1} g\left(\frac{k}{p+1}\right)$. □

Now let's prove Theorem 5. To prove that $\tau^* = \frac{r}{\beta}$ decreases with the increase of any p_i, $i \in [N]$, we just need to prove that β increases with the increase of any p_i. Note that

$$
\beta = \sum_{i=1}^{N} \frac{1}{\lambda_i} \left(1 - \frac{1}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i\left(\frac{\lambda_i p_i}{k} - \alpha_i\right)}\right) \tag{A.27}
$$

is dependent on both λ_i and p_i. In the following, we first show that the change of λ_i does not

impact β. Particularly, by taking the partial derivative of β with respect to λ_i, we have

$$\begin{aligned}\frac{\partial \beta}{\partial \lambda_i} =& -\frac{1}{\lambda_i^2}\left(1-\frac{1}{p_i}\sum_{k=1}^{p_i} e^{-\mu_i(\frac{\lambda_i p_i}{k}-\alpha_i)}\right) \\ &+\frac{1}{\lambda_i}\sum_{k=1}^{p_i}\frac{\mu_i}{k}e^{-\mu_i(\frac{\lambda_i p_i}{k}-\alpha_i)} \\ =& -\frac{1}{\lambda_i^2}+\frac{1}{\lambda_i^2}\sum_{k=1}^{p_i}\left(\frac{1}{p_i}+\lambda_i\frac{\mu_i}{k}\right)e^{-\mu_i(\frac{\lambda_i p_i}{k}-\alpha_i)} \\ \overset{(a)}{=}& -\frac{1}{\lambda_i^2}+\frac{1}{\lambda_i^2} \\ =&0, \end{aligned} \tag{A.28}$$

where (a) is obtained using Eq. (7). Therefore, β does not change as λ_i varies.

Next, we prove that β increases with the increase of any p_i, $i \in [N]$. Define the following auxiliary function for each $i \in [N]$,

$$g_i(x) = e^{-\frac{\mu_i \lambda_i}{x}} \tag{A.29}$$

We can easily verify that $g_i(x)$ is a concave and increasing function, as $\mu_i > 0, \lambda_i > 0$, and $g_i(x) \geq 0$. According to Lemma 10, $\frac{1}{p_i}\sum_{k=1}^{p_i} g_i\left(\frac{k}{p_i}\right) \geq \frac{1}{p_i+1}\sum_{k=1}^{p_i+1} g_i\left(\frac{k}{p_i+1}\right)$. Therefore, with the increase of any p_i, the term $\frac{1}{p_i}\sum_{k=1}^{p_i} e^{-\mu_i(\frac{\lambda_i p_i}{k}-\alpha_i)}$ in Eq. (A.27) decreases, causing β to increase.

A.4 Proof of Theorem 6

According to Theorem 5, τ^* decreases with the increase of any p_i, $i \in [N]$. Therefore, the infimum of τ^* is attained when $p_i \to \infty$, $\forall i \in [N]$, i.e.,

$$\inf \tau^* = \lim_{p_i \to \infty, \forall i \in [N]} \tau^* = \lim_{p_i \to \infty, \forall i \in [N]} \frac{r}{\beta} \tag{A.30}$$

Let's now calculate $\lim_{p_i \to \infty, \forall i \in [N]} \beta$. According to Lemma 1, we have $\lim_{p_i \to \infty} \lambda_i = \alpha_i$, which leads to

$$
\begin{aligned}
& \lim_{p_i \to \infty, \forall i \in [N]} \beta \\
= & \lim_{p_i \to \infty, \forall i \in [N]} \sum_{i=1}^{N} \frac{1}{\lambda_i} \left(1 - \frac{1}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i (\frac{\lambda_i p_i}{k} - \alpha_i)} \right) \\
= & \lim_{p_i \to \infty, \forall i \in [N]} \sum_{i=1}^{N} \frac{1}{\lambda_i} \left(1 - e^{\mu_i \alpha_i} \int_0^1 e^{-\frac{\mu_i \lambda_i}{x}} dx \right) \\
= & \sum_{i=1}^{N} \frac{1}{\alpha_i} (1 - e^{\mu_i \alpha_i} \int_0^1 e^{-\frac{\mu_i \alpha_i}{x}} dx)
\end{aligned}
\tag{A.31}
$$

Therefore,

$$
\begin{aligned}
\inf \tau^* = & \lim_{p_i \to \infty, \forall i \in [N]} \tau^* \\
= & \frac{r}{\sum_{i=1}^{N} \frac{1}{\alpha_i} (1 - e^{\mu_i \alpha_i} \int_0^1 e^{-\frac{\mu_i \alpha_i}{x}} dx)}
\end{aligned}
\tag{A.32}
$$

Similarly, from Theorem 5, we can derive that the supremum of τ^* is attained when $p_i = 1, \forall i \in [N]$, i.e.,

$$
\sup \tau^* = \sum_{i=1}^{N} \frac{1}{\sup \lambda_i} \left(1 - e^{-\mu_i (\sup \lambda_i - \alpha_i)} \right), \tag{A.33}
$$

where $\sup \lambda_i$ is given by Eq. (9).

A.5 Proof of Corollary 6.1

As τ^* approaches its infimum when $p_i \to \infty$, $\forall i \in [N]$, we have

$$\begin{aligned}
\hat{\ell}_i &= \lim_{p_j \to \infty, \forall j \in [N]} \ell_i^* \\
&= \lim_{p_j \to \infty, \forall j \in [N]} \frac{r}{\beta \lambda_i} \\
&= \frac{r}{\alpha_i \sum_{j=1}^{N} \frac{1}{\alpha_j} (1 - e^{\mu_j \alpha_j} \int_0^1 e^{-\frac{\mu_j \alpha_j}{x}} dx)},
\end{aligned} \tag{A.34}$$

according to Eq. (14) and Eq. (A.31) in the proof of Theorem 6.

A.6 Proof of Theorem 7

According to Theorem 4, we have $\lim_{N\to\infty} \mathbb{E}[T_{\text{BPCC}}] = \tau^*$. Similarly, according to [25], we can derive $\lim_{N\to\infty} \mathbb{E}[T_{\text{HCMM}}] = \tau_H^*$. Since HCMM is a special case of BPCC with $p_i = 1$, $\forall i \in [N]$, by applying Theorem 5, we have

$$\lim_{N\to\infty} \mathbb{E}[T_{\text{BPCC}}] \leq \lim_{N\to\infty} \mathbb{E}[T_{\text{HCMM}}]$$

Appendix B

Proofs of Chapter 5

B.1 Proof of Lemma 9

Consider the Q-value function Q_i^μ of agent i. For two different sets of non-neighbor states $\hat{\mathbf{s}}_{\mathcal{N}_i^-} \neq \mathbf{s}_{\mathcal{N}_i^-}$ and actions $\hat{\mathbf{a}}_{\mathcal{N}_i^-} \neq \mathbf{a}_{\mathcal{N}_i^-}$, we first show that:

$$\begin{aligned} &|Q_i^\mu(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-}) - Q_i^\mu(\mathbf{s}_{\mathcal{N}_i}, \hat{\mathbf{s}}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \hat{\mathbf{a}}_{\mathcal{N}_i^-})| \\ &\qquad \le \frac{2\bar{r}\gamma}{1-\gamma}. \end{aligned} \tag{B.1}$$

Letting $(\mathbf{s}, \mathbf{a})$ and $(\hat{\mathbf{s}}, \hat{\mathbf{a}})$ denote $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i^-})$ and $(\mathbf{s}_{\mathcal{N}_i}, \hat{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i}, \hat{a}_{\mathcal{N}_i^-})$, respectively, we have:

$$\begin{aligned} &\left|Q_i^\mu(\mathbf{s}, \mathbf{a}) - Q_i^\mu(\hat{\mathbf{s}}, \hat{\mathbf{a}})\right| \\ &= \Bigg|\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_i\left(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)\right) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})] \\ &- \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_i\left(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)\right) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})]\Bigg| \\ &\le \sum_{t=0}^{\infty} \Big|\mathbb{E}\left[\gamma^t r_i\left(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)\right) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})\right] \\ &- \mathbb{E}\left[\gamma^t r_i\left(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)\right) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})\right]\Big| \\ &\overset{(\mathbf{a})}{=} \sum_{t=1}^{\infty} \Big|\mathbb{E}\left[\gamma^t r_i\left(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)\right) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})\right] \end{aligned}$$

$$
\begin{aligned}
&- \mathbb{E}\left[\gamma^t r_i\left(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)\right) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})\right]\Big| \\
&\leq \sum_{t=1}^{\infty} \gamma^t \Big(\Big| \mathbb{E}\left[r_i\left(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)\right) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\mathbf{s}, \mathbf{a})\right]\Big| \\
&+ \Big| \mathbb{E}\left[r_i\left(\mathbf{s}_{\mathcal{N}_i}(t), \mathbf{a}_{\mathcal{N}_i}(t)\right) \mid (\mathbf{s}(0), \mathbf{a}(0)) = (\hat{\mathbf{s}}, \hat{\mathbf{a}})\right]\Big| \Big) \\
&\leq \sum_{t=1}^{\infty} 2\gamma^t \bar{r} = \frac{2\bar{r}\gamma}{1-\gamma}
\end{aligned} \tag{B.2}
$$

where $(\mathbf{a})$ derives from the fact that $(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i})$ are part of both $(\mathbf{s}, \mathbf{a})$ and $(\hat{\mathbf{s}}, \hat{\mathbf{a}})$. In the above equations, the expectation $\mathbb{E}$ is over state-action trajectories generated by the policy μ and the transition model p. Then, we have:

$$
\begin{aligned}
&\left|\tilde{Q}_i^{\mu}\left(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}\right) - Q_i^{\mu}(\mathbf{s}, \mathbf{a})\right| \\
&= \Bigg| \sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} \omega_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) Q_i^{\mu}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) \\
&\quad - Q_i^{\mu}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \hat{\mathbf{s}}_{\mathcal{N}_i^-}, \hat{\mathbf{a}}_{\mathcal{N}_i^-}) \Bigg| \\
&\leq \sum_{\mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}} \omega_i(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) \Big| Q_i^{\mu}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \mathbf{s}_{\mathcal{N}_i^-}, \mathbf{a}_{\mathcal{N}_i^-}) \\
&\quad - Q_i^{\mu}(\mathbf{s}_{\mathcal{N}_i}, \mathbf{a}_{\mathcal{N}_i}, \hat{\mathbf{s}}_{\mathcal{N}_i^-}, \hat{\mathbf{a}}_{\mathcal{N}_i^-}) \Big| \leq \frac{2\bar{r}\gamma}{1-\gamma}
\end{aligned}
$$

□

B.2 Proof of Proposition 1

If agent $j \notin \mathscr{P}_i(t)$, then based on the definition of potential neighbors, we have $\text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t)) > d + 2\varepsilon$. According to the triangle inequality, $\text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t+1)) + \text{dist}(\mathbf{s}_j(t+1), \mathbf{s}_j(t)) \geq \text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t))$, and according to Assumption 1, $\text{dist}(\mathbf{s}_j(t+1), \mathbf{s}_j(t)) \leq \varepsilon$. Therefore, $\text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t+1)) > d + \varepsilon$. Using the triangle inequality again, we obtain $\text{dist}(\mathbf{s}_i(t+1), \mathbf{s}_j(t+1)) + \text{dist}(\mathbf{s}_i(t+1), \mathbf{s}_i(t)) \geq \text{dist}(\mathbf{s}_i(t), \mathbf{s}_j(t+1))$. As $\text{dist}(\mathbf{s}_i(t+1), \mathbf{s}_i(t)) \leq \varepsilon$, we have $\text{dist}(\mathbf{s}_i(t+1), \mathbf{s}_j(t+1)) > d$. Therefore, agent j will not be a one-hop neighbor of agent i at time $t+1$. □

B.3 Proof of Proposition 3

The performance of the MDS code scheme will be affected only if the number of stragglers exceeds $N-M$ because $\mathbf{C}^{\mathrm{MDS}}$ has rank M. If there are $W > N-M$ stragglers, the results from non-straggler nodes will be insufficient for the central controller to decode the parameter gradients and it needs to wait for results from the stragglers. Under Assumption 4, W follows a binomial distribution with probability mass function $p(W=w) = \binom{N}{w}(1-\eta)^{N-w}\eta^{w}$. Therefore, the probability that the performance will be affected by the stragglers is $\sum_{j=N-M+1}^{N} p(W=j) = \sum_{j=N-M+1}^{N} \binom{N}{j}(1-\eta)^{N-j}\eta^{j}$.

Bibliography

[1] D. W. Casbeer, R. W. Beard, T. W. McLain, S.-M. Li, and R. K. Mehra, "Forest fire monitoring with multiple small uavs," in *Proceedings of the 2005 American Control Conference*, June Hilton Portland Portland, Oregon, June 2005, pp. 3530-3535.

[2] E. Kuiper and S. Nadjm-Tehrani, "Mobility models for uav group reconnaissance applications," in *Proceedings of the 2006 International Conference on Wireless and Mobile Communications*, July Vancouver, Canada, July 2006, pp. 33-33.

[3] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue," *IEEE robotics & automation magazine*, vol. 19, no. 3, pp. 46–56, 2012.

[4] F. Nex and F. Remondino, "Uav for 3d mapping applications: a review," *Applied geomatics*, vol. 6, no. 1, pp. 1–15, 2014.

[5] V. Hassija, V. Chamola, A. Agrawal, A. Goyal, N. C. Luong, D. Niyato, F. R. Yu, and M. Guizani, "Fast, reliable, and secure drone communication: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2021.

[6] H. Wang, H. Zhao, J. Zhang, D. Ma, J. Li, and J. Wei, "Survey on unmanned aerial vehicle networks: A cyber physical system perspective," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1027–1070, 2019.

[7] B. Alzahrani, O. S. Oubbati, A. Barnawi, M. Atiquzzaman, and D. Alghazzawi, "Uav assistance paradigm: State-of-the-art in applications and challenges," *Journal of Network and Computer Applications*, vol. 166, p. 102706, 2020.

[8] W. Chen, B. Liu, H. Huang, S. Guo, and Z. Zheng, "When uav swarm meets edge-cloud computing: The qos perspective," *IEEE Network*, vol. 33, no. 2, pp. 36–43, 2019.

[9] K. Lu, J. Xie, Y. Wan, and S. Fu, "Toward uav-based airborne computing," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 172–179, 2019.

[10] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[11] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, and G. Y. Li, "Joint offloading and trajectory design for uav-enabled mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1879–1892, 2018.

[12] F. Zhou, Y. Wu, R. Q. Hu, and Y. Qian, "Computation rate maximization in uav-enabled wireless-powered mobile-edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 1927–1941, 2018.

[13] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-uav-enabled load-balance mobile-edge computing for iot networks," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6898–6908, 2020.

[14] S. Jeong, O. Simeone, and J. Kang, "Mobile cloud computing with a uav-mounted cloudlet: optimal bit allocation for communication and computation," *Iet Communications*, vol. 11, no. 7, pp. 969–974, 2017.

[15] L. Lyu, F. Zeng, Z. Xiao, C. Zhang, H. Jiang, and V. Havyarimana, "Computation bits maximization in uav-enabled mobile edge computing system," *IEEE Internet of Things Journal*, 2021.

[16] M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao, and X. Shen, "Energy-efficient uav-assisted mobile edge computing: Resource allocation and trajectory optimization," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3424–3438, 2020.

[17] X. Hu, K.-K. Wong, K. Yang, and Z. Zheng, "Uav-assisted relaying and edge computing: Scheduling and trajectory optimization," *IEEE Transactions on Wireless Communications*, vol. 18, no. 10, pp. 4738–4752, 2019.

[18] J. Xiong, H. Guo, and J. Liu, "Task offloading in uav-aided edge computing: Bit allocation and trajectory optimization," *IEEE Communications Letters*, vol. 23, no. 3, pp. 538–541, 2019.

[19] A. Asheralieva and D. Niyato, "Hierarchical game-theoretic and reinforcement learning framework for computational offloading in uav-enabled mobile edge computing networks with multiple service providers," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8753–8769, 2019.

[20] H. Wang, H. Ke, and W. Sun, "Unmanned-aerial-vehicle-assisted computation offloading for mobile edge computing based on deep reinforcement learning," *IEEE Access*, vol. 8, pp. 180 784–180 798, 2020.

[21] H. Zhou, Z. Wang, G. Min, and H. Zhang, "Uav-aided computation offloading in mobile edge computing networks: a stackelberg game approach," *IEEE Internet of Things Journal*, 2022.

[22] H. Chang, Y. Chen, B. Zhang, and D. Doermann, "Multi-uav mobile edge computing and path planning platform based on reinforcement learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2021.

[23] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, and L. Hanzo, "Multi-agent deep reinforcement learning-based trajectory planning for multi-uav assisted mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 1, pp. 73–84, 2020.

[24] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *Proceedings of the 2017 IEEE ISIT*, Aachen, Germany, June 2017.

[25] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[26] ROS, "Robot operating system (ros)," Accessed: March 23, 2023. [Online]. Available: https://www.ros.org/

[27] GAZEBO, "Gazebo," Accessed: March 23, 2023. [Online]. Available: https://gazebosim.org/home

[28] A. C. Satici, H. Poonawala, and M. W. Spong, "Robust optimal control of quadrotor uavs," *IEEE Access*, vol. 1, pp. 79–93, 2013.

[29] A. Fotouhi, M. Ding, and M. Hassan, "Flying drone base stations for macro hotspots," *IEEE Access*, vol. 6, pp. 19 530–19 539, 2018.

[30] K. Li, W. Ni, X. Wang, R. P. Liu, S. S. Kanhere, and S. Jha, "Energy-efficient cooperative relaying for unmanned aerial vehicles," *IEEE Transactions on Mobile Computing*, vol. 15, no. 6, pp. 1377–1386, 2016.

[31] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Mobile internet of things: Can uavs provide an energy-efficient mobile architecture?" in *Proceedings of 2016 Global Communications Conference (GLOBECOM)*, December Washington D.C., USA, December 2016, pp. 1-6.

[32] Z. M. Fadlullah, D. Takaishi, H. Nishiyama, N. Kato, and R. Miura, "A dynamic trajectory control algorithm for improving the communication throughput and delay in uav-aided networks," *IEEE Network*, vol. 30, no. 1, pp. 100–105, 2016.

[33] N. H. Motlagh, M. Bagaa, and T. Taleb, "Uav-based iot platform: A crowd surveillance use case," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 128–134, 2017.

[34] B. Qureshi, A. Koubaa, M.-F. Sriti, Y. Javed, and M. Alajlan, "Poster: Dronemap-a cloud-based architecture for the internet-of-drones." in *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, Graz, Austria, 2016 February, pp. 255-256.

[35] A. Giyenko and Y. Im Cho, "Intelligent uav in smart cities using iot," in *Proceedings of 2016 16th International Conference on Control, Automation and Systems (ICCAS),*, October Gyeongju, Korea, 2016, pp. 207-210.

[36] S.-J. Yoo, J.-h. Park, S.-h. Kim, and A. Shrestha, "Flying path optimization in uav-assisted iot sensor networks," *ICT Express*, vol. 2, no. 3, pp. 140–144, 2016.

[37] "Jetson TX2 Module," Accessed: March 23, 2023. [Online]. Available: https://elinux.org/Jetson_TX2

[38] C. Dall and J. Nieh, "Kvm/arm: the design and implementation of the linux arm hypervisor," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 333–348, 2014.

[39] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[40] "Raspberry Pi," Accessed: March 23, 2023. [Online]. Available: https://www.raspberrypi.org/

[41] "Odroid Xu," Accessed: March 23, 2023. [Online]. Available: https://www.hardkernel.com/ko/tag/odroid-xu/

[42] "Arduino," Accessed: March 23, 2023. [Online]. Available: https://www.arduino.cc/

[43] "Cubieboard," Accessed: March 23, 2023. [Online]. Available: http://docs.cubieboard.org/products/start

[44] "Arndale Board," Accessed: March 23, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Arndale_Board

[45] Z. Shang and Z. Shen, "Real-time 3d reconstruction on construction site using visual slam and uav," *arXiv preprint arXiv:1712.07122*, 2017.

[46] "Jetson TX1 Module," Accessed: March 23, 2023. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx1

[47] "UDOO X86," Accessed: March 23, 2023. [Online]. Available: https://www.udoo.org/docs-x86/Introduction/Introduction.html

[48] "Intel Aero Compute Board," Accessed: March 23, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/97178/intel-aero-compute-board/specifications.html

[49] "LattePanda Alpha," Accessed: March 23, 2023. [Online]. Available: https://www.kickstarter.com/projects/139108638/lattepanda-alpha-soul-of-a-macbook-in-a-pocket-siz

[50] "UP Squared," Accessed: March 23, 2023. [Online]. Available: https://up-board.org/upsquared/specifications/

[51] "Raspberry Pi 4," Accessed: March 23, 2023. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

[52] "Jetson AGX Xavier," Accessed: March 23, 2023. [Online]. Available: https://www.nvidia.com/en-sg/autonomous-machines/embedded-systems/jetson-agx-xavier/#jetson-agx-xavier

[53] "DJI Manifold," Accessed: March 23, 2023. [Online]. Available: https://www.dji.com/manifold

[54] "HiKey960," Accessed: March 23, 2023. [Online]. Available: https://www.96boards.org/product/hikey960/

[55] "Rock 960," Accessed: March 23, 2023. [Online]. Available: https://www.96rocks.com/

[56] "Support Resources," Accessed: March 23, 2023. [Online]. Available: https://developer.nvidia.com/embedded/community/support-resources

[57] S. Li, C. He, M. Liu, Y. Wan, Y. Gu, J. Xie, S. Fu, and K. Lu, "Design and implementation of aerial communication using directional antennas: learning control in unknown communication environments," *IET Control Theory & Applications*, vol. 13, no. 17, pp. 2906–2916, 2019.

[58] "DJI Matrice 100," Accessed: March 23, 2023. [Online]. Available: https://www.dji.com/matrice100

[59] "NanoStation Loco M5," Accessed: March 23, 2023. [Online]. Available: https://store.ui.com/collections/wireless/products/nanolocom5

[60] "WS323 300Mbps Wireless Range Extender User Guide," Accessed: March 23, 2023. [Online]. Available: https://www.manualslib.com/manual/547195/Huawei-Ws323.html#manual

[61] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1, Ottawa, Ontario, July 2007.

[62] M.-K. Yoon, B. Liu, N. Hovakimyan, and L. Sha, "Virtualdrone: virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems," in *Proceedings of the 8th International Conference on Cyber-Physical Systems*, April Pittsburgh, PA, April 2017, pp. 143-154.

[63] N. Jain and S. Choudhary, "Overview of virtualization in cloud computing," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, Indore, India,March 2016, pp.1-4, pp. 1–4.

[64] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, and B.-J. Kim, "Performance comparison analysis of linux container and virtual machine for building cloud," *Advanced Science and Technology Letters*, vol. 66, no. 2, pp. 105–111, 2014.

[65] L. Malhotra, D. Agarwal, and A. Jaiswal, "Virtualization in cloud computing," *J Inform Tech Softw Eng*, vol. 4, no. 136, p. 2, 2014.

[66] J. Shuja, A. Gani, K. Bilal, A. U. R. Khan, S. A. Madani, S. U. Khan, and A. Y. Zomaya, "A survey of mobile device virtualization: Taxonomy and state of the art," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, pp. 1–36, 2016.

[67] R. Morabito, "A performance evaluation of container technologies on internet of things devices," in *Proceedings of the 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April San Francisco, CA ,April 2016, pp.999-1000.

[68] A. Patel, M. Daftedar, M. Shalan, and M. W. El-Kharashi, "Embedded hypervisor xvisor: A comparative analysis," in *Proceedings of the 2015 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, Turku, Finland, March 2015, pp. 682-691.

[69] F. Gu, F. Hu, and H. Chen, "Real-time performance evaluation of linux arm virtualization," in *Proceedings of the 2nd International Conference on Energy Science and Applied Technology (ESAT 2015)*, August Wuhan, China, August 2015.

[70] S. Toumassian, R. Werner, and A. Sikora, "Performance measurements for hypervisors on embedded arm processors," in *Proceedings of 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, September Jaipur, India, September 2016, pp.851-858.

[71] M. Raho, A. Spyridakis, M. Paolino, and D. Raho, "Kvm, xen and docker: A performance analysis for arm based nfv and cloud computing," in *Proceedings of the 2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, November Riga, Latvia, November 2015, pp. 1-8.

[72] R. Morabito and N. Beijar, "Enabling data processing at the network edge through lightweight virtualization technologies," in *Proceedings of 2016 IEEE International Conference on Sensing, Communication and Networking (SECON Workshops)*, London, UK,June 2016, pp. 1-6.

[73] R. Morabito, "Virtualization on internet of things edge devices with container technologies: a performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

[74] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over raspberrypi," in *Proceedings of the 18th international conference on distributed computing and networking*, NY, USA, January 2017, p. 16.

[75] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study," *IEEE access*, vol. 5, pp. 9882–9910, 2017.

[76] "Edge Computing for UAVs, UASs, and Drones," Accessed: March 23, 2023. [Online]. Available: https://www.nutanix.com/go/edge-computing-for-drones.html

[77] N. Kalatzis, M. Avgeris, D. Dechouniotis, K. Papadakis-Vlachopapadopoulos, I. Roussaki, and S. Papavassiliou, "Edge computing in iot ecosystems for uav-enabled early fire detection," in *Proceedings of 2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, July Kuala Lumpur, Malaysia, July 2018, pp.106-114.

[78] B. Wang, J. Xie, S. Li, Y. Wan, S. Fu, and K. Lu, "Enabling high-performance onboard computing with virtualization for unmanned aerial systems," in *Proceedings of 2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, Dallas, TX, June 2018, pp. 202-211.

[79] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of IEEE International Symposium on Workload Characterization*, October Austin, TX, October 2009, pp.44-54.

[80] R. Montella, G. Giunta, G. Laccetti, M. Lapegna, C. Palmieri, C. Ferraro, and V. Pelliccia, "Virtualizing cuda enabled gpgpus on arm clusters," in *Parallel Processing and Applied Mathematics*. Springer, 2016, pp. 3–14.

[81] D. Kang, T. J. Jun, D. Kim, J. Kim, and D. Kim, "Convgpu: Gpu management middleware in container based virtualized environment," in *Proceedings of the 2017 IEEE International Conference on Cluster Computing (CLUSTER)*, September Hawaii, USA,September 2017, pp.301-309.

[82] A. Tirumala, T. Dunigan, and L. Cottrell, "Measuring end-to-end bandwidth with iperf using web100," in *Proceedings of Passive and Active Monitoring Workshop*, April San Diego, CA, April 2003.

[83] Z. Wei, G. Xiaolin, H. R. Wei, and Y. Si, "Tcp ddos attack detection on the host in the kvm virtual machine environment," in *Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science (ICIS)*, May Shanghai, China, May 2012, pp. 62-67.

[84] R. Mabry, J. Ardonne, J. N. Weaver, D. Lucas, and M. J. Bays, "Maritime autonomy in a box: Building a quickly-deployable autonomy solution using the docker container environment," in *Proceedings of OCEANS 2016 MTS/IEEE Monterey*, September Monterey, CA, September 2016, pp.1-6.

[85] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Proceedings of the 2013 21st Euromicro International Conference*

on Parallel, Distributed and Network-Based Processing (PDP), February Belfast, UK, February 2013, pp. 233-240.

[86] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *Proceedings of the 2007 workshop on Experimental computer science*, June San Diego, CA, June 2007, p.6.

[87] D. H. Bailey, *NAS Parallel Benchmarks*. Boston, MA: Springer US, 2011, pp. 1254–1259.

[88] "Jetson TX2 Thermal Design Guide," Accessed: March 23, 2023. [Online]. Available: https://devtalk.nvidia.com/default/topic/1036126/measure-jetson-x2-energy-usage-during-a-given-task/

[89] "Sysstat," Accessed: March 23, 2023. [Online]. Available: http://sebastien.godard.pagesperso-orange.fr/

[90] C. Yu and F. Huan, "Live migration of docker containers through logging and replay," in *Proceedings of the International Conference on Mechatronics and Industrial Informatics Advances in Computer Science Research*, October Zhuhai, China, October 2015.

[91] "Checkpoint and Restore," Accessed: March 23, 2023. [Online]. Available: https://criu.org/Main_Page

[92] "Docker," Accessed: March 23, 2023. [Online]. Available: https://www.docker.com/

[93] "OpenDroneMap," Accessed: March 23, 2023. [Online]. Available: http://opendronemap.org/

[94] "Jetson inference," Accessed: March 23, 2023. [Online]. Available: https://github.com/dusty-nv/jetson-inference

[95] "UCF Aerial Action Data Set," Accessed: March 23, 2023. [Online]. Available: http://crcv.ucf.edu/data/UCF_Aerial_Action.php

[96] S. Kartik and C. S. Ram Murthy, "Task allocation algorithms for maximizing reliability of distributed computing systems," *IEEE Transactions on Computers*, vol. 46, no. 6, pp. 719–724, 1997.

[97] B. Hong and V. K. Prasanna, "Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput," in *Proceedings of the 2004 IPDPS*, Santa Fe, New Mexico, April 2004.

[98] K. Lu, J. Xie, Y. Wan, and S. Fu, "Toward UAV-Based Airborne Computing," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 172–179, 2019.

[99] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[100] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2010 HotCloud*, Boston, USA, June 2010.

[101] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[102] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely-throughput optimal coded computing over cloud networks," in *Proceedings of the 20th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Catania, Italy, July 2019.

[103] K. T. Kim, C. Joe-Wong, and M. Chiang, "Coded edge computing," in *Proceedings of the 2020 IEEE INFOCOM*, Virtual, July 2020.

[104] J. Yue and M. Xiao, "Coding for distributed fog computing in internet of mobile things," *IEEE Transactions on Mobile Computing*, 2020.

[105] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *Proceedings of the 2008 OSDI*, San Diego, CA, December 2008.

[106] J. Dean, "Achieving Rapid Response Times in Large Online Services," Accessed: March 23, 2023. [Online]. Available: https://research.google/pubs/pub44875/

[107] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective Straggler Mitigation: Attack of the Clones," in *Proceedings of the 2013 NSDI*. Lombard, IL: USENIX, April 2013.

[108] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using Mantri," in *Proceedings of the 2010 OSDI*, BC, Canada, October 2010.

[109] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, pp. 330–339, 2010.

[110] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Why let resources idle? aggressive cloning of jobs with dolly," in *Proceedings of the 2012 HotCloud*. Boston, MA: USENIX, June 2012.

[111] C. Liu, Q. Wang, X. Chu, Y.-W. Leung, and H. Liu, "Esetstore: An erasure-coded storage system with fast data recovery," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2001–2016, 2020.

[112] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Proceedings of the 2015 Allerton*, IL, USA, September 2015.

[113] Y. Zhu, P. P. Lee, Y. Xu, Y. Hu, and L. Xiang, "On the speedup of recovery in large-scale erasure-coded storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1830–1840, 2013.

[114] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," in *Proc. of IEEE ISIT 2016*. IEEE, aug 2016, pp. 1143–1147.

[115] ——, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[116] N. S. Ferdinand and S. C. Draper, "Anytime coding for distributed computation," in *Proceedings of the 2016 Allerton*, IL, USA, September 2016.

[117] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," in *Proc. of the 34th International Conference on Machine Learning*. PMLR, 2017, pp. 3368–3376.

[118] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proceedings of the 2017 IEEE ISIT*, Aachen, Germany, June 2017.

[119] Y. Yang, P. Grover, and S. Kar, "Coded Distributed Computing for Inverse Problems," *Advances in Neural Information Processing Systems*, pp. 710–720, 2017.

[120] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *Proceedings of the 2017 IEEE ISIT*, Aachen, Germany, June 2017.

[121] S. Dutta, V. Cadambe, and P. Grover, "Short-Dot: Computing Large Linear Transforms Distributedly Using Coded Short Dot Products," *arXiv*, apr 2017. [Online]. Available: https://arxiv.org/abs/1704.05181

[122] S. Li, M. A. Maddah-Ali, and A. Salman Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proceedings of the IEEE Globecom 2016 Workshops*, Washington, DC USA, December 2016.

[123] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A Fundamental Tradeoff Between Computation and Communication in Distributed Computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, jan 2018.

[124] M. Kim, J.-Y. Sohn, and J. Moon, "Coded Matrix Multiplication on a Group-Based Model," *arXiv*, jan 2019. [Online]. Available: http://arxiv.org/abs/1901.05162

[125] D. Kim, H. Park, and J. Choi, "Optimal Load Allocation for Coded Distributed Computation in Heterogeneous Clusters," *arXiv*, 2019. [Online]. Available: http://arxiv.org/abs/1904.09496

[126] Y. Keshtkarjahromi, Y. Xing, and H. Seferoglu, "Dynamic heterogeneity-aware coded cooperative computation at the edge," in *Proceedings of the 2018 ICNP*, Athens, Greece, September 2018.

[127] K. G. Narra, Z. Lin, M. Kiamari, S. Avestimehr, and M. Annavaram, “Slack squeeze coded computing for adaptive straggler mitigation,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, November 2019.

[128] N. Ferdinand and S. C. Draper, “Hierarchical Coded Computation,” in *Proceedings of 2018 IEEE ISIT*, Vail, USA, June 2018.

[129] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, “Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–40, 2019.

[130] R. M. Corless, G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth, “On the lambertw function,” *Advances in Computational mathematics*, vol. 5, no. 1, pp. 329–359, 1996.

[131] AWS, “Amazon ec2,” Accessed: March 23, 2023. [Online]. Available: https://aws.amazon.com/ec2/

[132] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1999.

[133] “mpi4py,” Accessed: March 23, 2023. [Online]. Available: https://mpi4py.readthedocs.io/en/stable/

[134] J.-X. Pan and K.-T. Fang, “Maximum likelihood estimation,” in *Growth curve models and statistical diagnostics*. Springer, 2002, pp. 77–158.

[135] D. Sharma, “Estimation of the reciprocal of the scale parameter in a shifted exponential distribution,” *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 203–205, 1977.

[136] B. Wang, J. Xie, S. Li, Y. Wan, Y. Gu, S. Fu, and K. Lu, “Computing in the air: An open airborne computing platform,” *IET Communications*, vol. 14, no. 15, pp. 2410–2419, 2020.

[137] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, “Coding for heterogeneous uav-based networked airborne computing,” in *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019, pp. 1–6.

[138] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, and S. Guizani, “Mobile ad hoc cloud: A survey,” *Wireless Communications and Mobile Computing*, vol. 16, no. 16, pp. 2572–2589, 2016.

[139] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, and M. Imran, “Heterogeneity-aware task allocation in mobile ad hoc cloud,” *IEEE Access*, vol. 5, pp. 1779–1795, 2017.

[140] A. J. Ferrer, J. M. Marquès, and J. Jorba, “Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[141] B. Li, Y. Pei, H. Wu, and B. Shen, "Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds," *The Journal of Supercomputing*, vol. 71, no. 8, pp. 3009–3036, 2015.

[142] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "mcloud: A context-aware offloading framework for heterogeneous mobile cloud," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 797–810, 2015.

[143] N. Fernando, S. W. Loke, and W. Rahayu, "Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*. IEEE, 2011, pp. 281–286.

[144] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Networks and Applications*, pp. 1–24, 2020.

[145] W. Zhang, L. Li, N. Zhang, T. Han, and S. Wang, "Air-ground integrated mobile edge networks: A survey," *IEEE Access*, vol. 8, pp. 125 998–126 018, 2020.

[146] Y. Wang, Z.-Y. Ru, K. Wang, and P.-Q. Huang, "Joint deployment and task scheduling optimization for large-scale mobile users in multi-uav-enabled mobile edge computing," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3984–3997, 2019.

[147] Z. Yang, C. Pan, K. Wang, and M. Shikh-Bahaei, "Energy efficient resource allocation in uav-enabled mobile edge computing networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 9, pp. 4576–4589, 2019.

[148] Y. Luo, W. Ding, and B. Zhang, "Optimization of task scheduling and dynamic service strategy for multi-uav-enabled mobile edge computing system," *IEEE Transactions on Cognitive Communications and Networking*, 2021.

[149] Q. Liu, L. Shi, L. Sun, J. Li, M. Ding, and F. Shu, "Path planning for uav-mounted mobile edge computing with deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 5723–5728, 2020.

[150] X. Tao and W. Song, "Task allocation for mobile crowdsensing with deep reinforcement learning," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2020, pp. 1–7.

[151] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in iot edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1133–1146, 2020.

[152] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 102, pp. 847–861, 2020.

[153] O. S. Oubbati, M. Atiquzzaman, A. Baz, H. Alhakami, and J. Ben-Othman, “Dispatch of uavs for urban vehicular networks: A deep reinforcement learning approach,” *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13 174–13 189, 2021.

[154] O. S. Oubbati, A. Lakas, and M. Guizani, “Multi-agent deep reinforcement learning for wireless-powered uav networks,” *IEEE Internet of Things Journal*, 2022.

[155] Y. Keshtkarjahromi, Y. Xing, and H. Seferoglu, “Adaptive and heterogeneity-aware coded cooperative computation at the edge,” *IEEE Transactions on Mobile Computing*, 2021.

[156] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, “Coded computation over heterogeneous clusters,” *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[157] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, “On batch-processing based coded computing for heterogeneous distributed computing systems,” *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2438–2454, 2021.

[158] Y. Yang, P. Grover, and S. Kar, “Coded distributed computing for inverse problems,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 709–719.

[159] S. Dutta, V. Cadambe, and P. Grover, “Coded convolution for parallel and distributed computing within a deadline,” in *Proceedings of the 2017 IEEE ISIT*, Aachen, Germany, June 2017.

[160] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, “A unified coded deep neural network training strategy based on generalized polydot codes,” in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1585–1589.

[161] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “Coded MapReduce,” in *Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2015, pp. 964–971.

[162] B. Wang, J. Xie, and N. Atanasov, “Coding for distributed multi-agent reinforcement learning,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 10 625–10 631.

[163] A. Frigård, S. Kumar, E. Rosnes *et al.*, “Rateless codes for low-latency distributed inference in mobile edge computing,” *arXiv preprint arXiv:2108.07675*, 2021.

[164] A. Asheralieva and D. Niyato, “Fast and secure computational offloading with lagrange coded mobile edge computing,” *IEEE Transactions on Vehicular Technology*, vol. 70, no. 5, pp. 4924–4942, 2021.

[165] P. Brusilovski, A. Kobsa, and W. Nejdl, *The adaptive web: methods and strategies of web personalization*. Springer Science & Business Media, 2007, vol. 4321.

[166] Y. Liu, P. Sun, N. Wergeles, and Y. Shang, “A survey and performance evaluation of deep learning methods for small object detection,” *Expert Systems with Applications*, vol. 172, p. 114602, 2021.

[167] B. Zhou, J. Xie, and B. Wang, “Dynamic coded convolution with privacy awareness for mobile ad hoc computing,” in *2022 IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2022.

[168] C. Douma, B. Wang, and J. Xie, “Coded distributed path planning for unmanned aerial vehicles,” in *AIAA AVIATION 2021 FORUM*, 2021, p. 2378.

[169] Q. Y. Kenny *et al.*, “Indicator function and its application in two-level factorial designs,” *The Annals of Statistics*, vol. 31, no. 3, pp. 984–994, 2003.

[170] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[171] R. S. Sutton and A. G. Barto, “An introduction to reinforcement learning, chapter 3,” 2018.

[172] A. C. Marreiros, J. Daunizeau, S. J. Kiebel, and K. J. Friston, “Population dynamics: variance and the sigmoid activation function,” *Neuroimage*, vol. 42, no. 1, pp. 147–157, 2008.

[173] A. F. Agarap, “Deep learning using rectified linear units (relu),” *arXiv preprint arXiv:1803.08375*, 2018.

[174] A. Redder, A. Ramaswamy, and H. Karl, “Asymptotic convergence of deep multi-agent actor-critic algorithms,” *arXiv preprint arXiv:2201.00570*, 2022.

[175] C. Qiu, Y. Hu, Y. Chen, and B. Zeng, “Deep deterministic policy gradient (ddpg)-based energy harvesting wireless communications,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8577–8588, 2019.

[176] Y. Al-Eryani, M. Akrout, and E. Hossain, “Multiple access in cell-free networks: Outage performance, dynamic clustering, and deep reinforcement learning-based design,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 4, pp. 1028–1042, 2020.

[177] M. Liu, Y. Wan, S. Li, F. L. Lewis, and S. Fu, “Learning and uncertainty-exploited directional antenna control for robust long-distance and broad-band aerial communication,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 593–606, 2019.

[178] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” in *Advances in neural information processing systems*, 2017, pp. 6379–6390.

[179] Y. Jin, S. Wei, J. Yuan, and X. Zhang, “Hierarchical and stable multiagent reinforcement learning for cooperative navigation control,” *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[180] R. Song, F. L. Lewis, and Q. Wei, “Off-policy integral reinforcement learning method to solve nonlinear continuous-time multiplayer nonzero-sum games,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 704–713, 2016.

[181] G.-P. Antonio and C. Maria-Dolores, “Multi-agent deep reinforcement learning to manage connected autonomous vehicles at tomorrow’s intersections,” *IEEE Transactions on Vehicular Technology*, vol. 71, no. 7, pp. 7033–7043, 2022.

[182] L. Buşoniu, R. Babuška, and B. De Schutter, “Multi-agent reinforcement learning: An overview,” *Innovations in Multi-agent Systems and Applications*, pp. 183–221, 2010.

[183] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, “Counterfactual multi-agent policy gradients,” in *AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[184] G. Qu, A. Wierman, and N. Li, “Scalable reinforcement learning of localized policies for multi-agent networked systems,” in *Learning for Dynamics and Control (L4DC)*. PMLR, 2020, pp. 256–266.

[185] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, “Mean field multi-agent reinforcement learning,” in *International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 5571–5580.

[186] Q. Long, Z. Zhou, A. Gupta, F. Fang, Y. Wu, and X. Wang, “Evolutionary population curriculum for scaling multi-agent reinforcement learning,” in *International Conference on Learning Representations (ICLR)*, 2020.

[187] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, “More effective distributed ml via a stale synchronous parallel parameter server,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2013, pp. 1223–1231.

[188] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, “Communication efficient distributed machine learning with the parameter server,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 19–27.

[189] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, “Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd,” in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 803–812.

[190] OpenAI, “A2C,” Accessed: March 23, 2023. [Online]. Available: https://openai.com/blog/baselines-acktr-a2c/

[191] M. Tan, “Multi-agent reinforcement learning: Independent vs. cooperative agents,” in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[192] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, “Value-decomposition networks for cooperative multi-agent learning,” *arXiv preprint:1706.05296*, 2017.

[193] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, “Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning,” in *International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 4295–4304.

[194] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, “Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 5887–5896.

[195] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis, “Multiagent reinforcement learning for urban traffic control using coordination graphs,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2008, pp. 656–671.

[196] J. R. Kok and N. Vlassis, “Collaborative multiagent reinforcement learning by payoff propagation,” *Journal of Machine Learning Research*, vol. 7, pp. 1789–1828, 2006.

[197] W. Böhmer, V. Kurin, and S. Whiteson, “Deep coordination graphs,” in *International Conference on Machine Learning (ICML)*. PMLR, 2020, pp. 980–991.

[198] S. Iqbal and F. Sha, “Actor-attention-critic for multi-agent reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 2961–2970.

[199] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, “Massively parallel methods for deep reinforcement learning,” in *International Conference on Machine Learning (ICML)*, 2015.

[200] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International Conference on Machine Learning (ICML)*, 2016, pp. 1928–1937.

[201] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, “Reinforcement learning through asynchronous advantage actor-critic on a gpu,” in *International Conference on Learning Representations (ICLR)*, 2017.

[202] D. Simões, N. Lau, and L. P. Reis, “Multi-agent actor centralized-critic with communication,” *Neurocomputing*, 2020.

[203] T. Chen, K. Zhang, G. B. Giannakis, and T. Başar, “Communication-efficient policy gradient methods for distributed reinforcement learning,” *IEEE Transactions on Control of Network Systems*, vol. 9, no. 2, pp. 917–929, 2022.

[204] B. Zhou, J. Xie, and B. Wang, "Dynamic Coded Distributed Convolution for UAV-based Networked Airborne Computing," in *IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*, 2022, pp. 955–961.

[205] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[206] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *International Conference on Learning Representations (ICLR)*, 2016.

[207] F. Bullo, J. Cortés, and S. Martinez, *Distributed control of robotic networks: a mathematical approach to motion coordination algorithms*. Princeton University Press, 2009.

[208] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2019.

[209] Y. Lin, G. Qu, L. Huang, and A. Wierman, "Distributed reinforcement learning in multi-agent networked systems," *arXiv preprint:2006.06555*, 2020.

[210] J. Lacan and J. Fimes, "Systematic mds erasure codes based on vandermonde matrices," *IEEE Communications Letters*, vol. 8, no. 9, pp. 570–572, 2004.

[211] A. Klinger, "The vandermonde matrix," *The American Mathematical Monthly*, vol. 74, no. 5, pp. 571–574, 1967.

[212] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2413–2417.

[213] E. M. Gabidulin and M. Bossert, "On the rank of ldpc matrices constructed by vandermonde matrices and rs codes," in *International Symposium on Information Theory*, 2006, pp. 861–865.

[214] S. Cai, W. Lin, X. Yao, B. Wei, and X. Ma, "Systematic convolutional low density generator matrix code," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3752–3764, 2021.

[215] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint:1412.6980*, 2014.

[216] A. Douklias, L. Karagiannidis, F. Misichroni, and A. Amditis, "Design and implementation of a uav-based airborne computing platform for computer vision and machine learning applications," *Sensors*, vol. 22, no. 5, p. 2049, 2022.

[217] J. Diller, P. Hall, C. Schanker, K. Ung, P. Belous, P. Russell, and Q. Han, "Iccswarm: A framework for integrated communication and control in uav swarms," in *Proceedings of the Eighth Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, 2022, pp. 1–6.

[218] E. Pereira, K. Hedrick, and R. Sengupta, "The c3uv testbed for collaborative control and information acquisition using uavs," in *2013 American Control Conference*. IEEE, 2013, pp. 1466–1471.

[219] M. Schmittle, A. Lukina, L. Vacek, J. Das, C. P. Buskirk, S. Rees, J. Sztipanovits, R. Grosu, and V. Kumar, "Openuav: A uav testbed for the cps and robotics community," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 130–139.

[220] M. Moradi, K. Sundaresan, E. Chai, S. Rangarajan, and Z. M. Mao, "Skycore: Moving core to the edge for untethered and reliable uav-based lte networks," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 35–49.

[221] S. Li, C. He, M. Liu, Y. Wan, Y. Gu, J. Xie, S. Fu, and K. Lu, "Design and implementation of aerial communication using directional antennas: learning control in unknown communication environments," *IET Control Theory & Applications*, vol. 13, no. 17, pp. 2906–2916, 2019.

[222] R. K. Sheshadri, E. Chai, K. Sundaresan, and S. Rangarajan, "Skyhaul: An autonomous gigabit network fabric in the sky," *arXiv preprint arXiv:2006.11307*, 2020.

[223] A. Y. Javaid, W. Sun, and M. Alam, "Uavsim: A simulation testbed for unmanned aerial vehicle network cyber security analysis," in *2013 ieee globecom workshops (gc wkshps)*. IEEE, 2013, pp. 1432–1436.

[224] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

[225] C. Wu and J. Z. Yu, "Evaluation of linear regression techniques for atmospheric applications: the importance of appropriate weighting," *Atmospheric Measurement Techniques*, vol. 11, no. 2, pp. 1233–1250, 2018.

[226] U. M. L. Repository, "Forest fires data set," Accessed: March 23, 2023. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/forest+fires

[227] ROS, "Ros topic," Accessed: March 23, 2023. [Online]. Available: http://wiki.ros.org/Topics

[228] S. Moon, J. J. Bird, S. Borenstein, and E. W. Frew, "A gazebo/ros-based communication-realistic simulator for networked suas," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2020, pp. 1819–1827.

[229] Rviz, "Rviz," Accessed: March 23, 2023. [Online]. Available: http://wiki.ros.org/rviz

[230] M. Hata, "Empirical formula for propagation loss in land mobile radio services," *IEEE transactions on Vehicular Technology*, vol. 29, no. 3, pp. 317–325, 1980.

[231] Y. Okumura, "Field strength and its variability in vhf and uhf land-mobile radio service," *Rev. Electr. Commun. Lab.*, vol. 16, pp. 825–873, 1968.

[232] "MPI," Accessed: March 23, 2023. [Online]. Available: https://ds.cs.luc.edu/mpi/mpi.html

[233] "Jetson TX2," Accessed: March 23, 2023. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/

[234] "Q Ground Control," Accessed: March 23, 2023. [Online]. Available: http://qgroundcontrol.com/

[235] "Iperf 3," Accessed: March 23, 2023. [Online]. Available: https://iperf.fr/iperf-download.php

[236] W. Rudin *et al.*, *Principles of mathematical analysis*. McGraw-hill New York, 1964, vol. 3.

[237] R. Combes, "An extension of McDiarmid's inequality," *arXiv*, 2015. [Online]. Available: http://arxiv.org/abs/1511.05240