TACTICAL HPC: SCHEDULING HIGH PERFORMANCE COMPUTERS IN A GEOGRAPHICAL REGION

A Thesis
Presented to
The Academic Faculty

by

Alireza K. Monfared

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2016

TACTICAL HPC: SCHEDULING HIGH PERFORMANCE COMPUTERS IN A GEOGRAPHICAL REGION

Approved by:

Professor Mary Ann Weitnauer,
Committee Chair
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor George Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Ellen W. Zegura, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Jim Xu
School of Computer Science
Georgia Institute of Technology

Professor Mostafa H. Ammar, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. David Doria
Computational and Information Sciences Directorate (CISD)
United States Army Research Laboratories

Date Approved: 4 December 2015

To my parents,

who sacrificed their happiness to support my success.

PREFACE

This work was supported in part by the US National Science Foundation through grant CNS 1161879 and United States Army Research Laboratory through grant No. 36566CB.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisors, Professor Ellen W. Zegura and Professor Mostafa H. Ammar for their guidance, understanding, patience, and most importantly, their unconditional support during my graduate studies at Georgia Tech. I would appreciate the input and careful feedback of my committee members, Dr. George Riley, my respected Academic Advisor, Dr. Mary Ann Weitnauer, my respected committee chair, Dr. Jim Xu, and last but not least, Dr. David Doria who has been a great colleague and friend throughout this journey and has provided utterly constructive feedback on my work at numerous occasions.

I would also like to extend my appreciation to all the academic members of the School of Electrical and Computer Engineering as well as College of Computing at the Georgia Institute of Technology for their helpful and critical reviews throughout the Ph.D. program. In specific, I have to extend my gratitude to Ms. Daniela Staiculescu who has given me guidance on the logistics of the defense and has been a great guide to help me with the formalities of completing a Ph.D. I would also like to thank all my colleagues at Network Research Group in Georgia Tech who have always given me great feedback.

On the non-academic front, I would like to extend my sincere thanks to my roommate for five years, Dr. Nassir Mokarram, who has also been like a brother to me, and to Izadi family, who have been like my family away from home. I would also like to thank my girlfriend, Parnia, for making the last part of this adventure even more exciting. And finally, my last and largest thanks shall go to my family, to my most loving father and mother and sister, that without slightest doubt, whatever that I have achieved during the past 29 years of my life, is the fruit of their love and

sacrifice.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Mobile devices are often expected to perform computational tasks that may be beyond their processing or battery capability. Cloud computing techniques have been proposed as a means to offload a mobile device's computation to more powerful resources. In this thesis, we consider the case where powerful computing resources are made available by utilizing vehicles. These vehicles can be repositioned in real time to receive computational tasks from user-carried devices. They can be either equipped with rugged high-performance computers to provide both computation and communication service, or they can be simple message ferries that facilitate communication with a more powerful computing resource. These scenarios find application in challenged environments and may be used in a military or disaster relief settings. It is further enabled by increasing feasibility of (i) constructing a Mobile High Performance Computer (MHPC) using rugged computer hardware with form factors that can be deployed in vehicles and (ii) Message Ferries (MF) that provide communication service in disruption tolerant networks. By analogy to prior work on message ferries and data mules, one can refer to the use of our first schema, MHPCs, as *computational ferrying*. After illustrating and motivating the computational ferrying concept, we turn our attention into the challenges facing such a deployment. These include the well-known challenges of operating an opportunistic and intermittently connected network using message ferries – such as devising an efficient mobility plan for MHPCs and developing techniques for proximity awareness. In this thesis, first we propose an architecture for the system components to be deployed on the mobile devices and the MHPCs. We then focus on defining and solving the MHPC movement scheduling

problem with sufficient generality to describe a number of plausible deployment scenarios. After thorough examination of the MHPC concepts, we propose a scheme in which MHPCs are downgraded to be simple MFs that instead provide communication to a stationary HPC with powerful computing resources. Similar to the MPHCs, we provide a framework for this problem and then describe heuristics to solve it. We conduct a number of experiments that provide an understanding of how the performance of the system using MHPCs or MFs is affected by various parameters. We also provide a thorough comparison of the system in the dimensions of Computation on the Move and Controlling the Mobility.

CHAPTER I

BACKGROUND AND LITERATURE OVERVIEW

1.1 Introduction

As computing devices become more powerful, expectations for performance and richness of applications rise as well. This creates an ever-present cycle of software exceeding current hardware capabilities and driving future development. This trend applies across many computing device types but is especially acute for mobile wireless devices, which are increasingly expected to perform demanding computational tasks. However, mobile devices have capability limits, battery lifetime limits, and compute power limits that restrict their ability to perform intensive computational tasks. This gap between expectations and mobile device capability is likely to continue.

In this thesis, we consider two ways to deploy powerful computing resources locally to enable mobile devices to accomplish what is asked of them. In the first case, resources are deployed on a vehicle, thus they can be repositioned in real-time. In the second case, resources are deployed in a fixed location with vehicles providing connectivity to this location. Mobile devices initiate computing tasks that they desire to run remotely, yet cannot reliably run on a remote cloud via the Internet. This scenario is found in challenged environments such as military or disaster relief efforts. It is further enabled by the increasing feasibility of constructing a (i) mobile high performance computer (MHPC) using rugged computing hardware with form factors that can be deployed in vehicles and (ii) also by application of a Message Ferry (MF) and Tactical High Performance Cloudlets (HPC). While expectations in challenged environments are often tempered, low-latency, computationally intensive applications can be important. The United States Army has research interests in this direction,

(a) HPC mounted on a military vehicle. **(b)** A sample HPC module, TSY-300X 3U VPX by Themis Computers [20].

Figure 1: Examples of ultra-rugged, battle-proven, high-performance computers used as HPCs in battlefield and disaster relief scenarios. We consider mounting these HPCs on vehicles and communicating with user nodes directly (1a) and also as stationary entities communicating with user nodes via Message Ferries.

under the label of Tactical High Performance Computing[35, 46, 50]. Figures 1a and 1b show examples of tactical HPCs used in combat vehicles.

In both of the above problems, there exists a request for assistance with computationally intensive tasks. The goal is to provide assistance with processing of the tasks. These tasks are initiated by user nodes that can be soldiers in a combat field or social workers in a disaster relief setting. The user nodes are geographically dispersed in the area. In the first problem, which we call the MHPC problem, we provide the assistance via the MHPCs that can go to each user node and receive tasks from them, process these tasks and bring back the results to the user node at a later time. In the second problem, which we call the HPC+MF problem, the computation resource is placed at a fixed location rather than being mounted on the vehicles. In this scenario, MFs provide communication for the user nodes and the HPC by picking up the tasks and dropping them off at the HPC, as well as receiving the processed results from the HPC and delivering them to the user nodes. The HPC then focuses on the processing

of these tasks.

In this thesis, we attempt to address various questions regarding the applicability of the MHPCs or the combination of the HPC and MFs to serve the computationally intensive tasks of the user nodes. We will review similar problems addressed in literature in the next section.

1.2 Literature Review

Our work draws on and complements work in mobile cloud computing, vehicular cloud computing, and disruption tolerant networks as well as tactical high performance computing trend of work in the military. In this section, we will summarize and discuss relevant work from these fields.

1.2.1 Cloud Computing and Mobile Cloud Computing

Cloud computing (CC) is defined as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." by The National Institute of Standards and Technology (NIST) [32]. In a high-level view, cloud computing techniques propose a means to offload intensive computational tasks from a mobile device to more powerful computing resources reached over the Internet or a similar network. Note that the term Cloud Computing can either refer to the services provided by applications that run over these networks or the actual hardware and architecture that provides such services [4].

Mobile Cloud Computing (MCC) brings the CC paradigm to the world of non-stationary devices. MCC is formally defined as "an infrastructure where both the data storage and data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and MCC to not just smartphone users but a

much broader range of mobile subscribers" [16]. Many forms of data processing have been shown to be advantageous if performed remotely, including image processing, natural language processing, crowd computing, sharing GPS/Internet data, sensor data applications, multimedia search, and social networking [18]. In [15] Cuervo et al. propose a system to offload fine-grained pieces of code that are chosen in an online manner in order to maximize the energy savings. This is an example of how MCC can be used to help with heavy computational tasks and save energy while minimally burdening the programmer of the original applications. In their evaluations, the authors apply MCC to face recognition and mobile games. Another example of automation of offloading is proposed by Chun et al.[13] where applications are partitioned at runtime and offloaded. Their evaluations shows up to 20x execution speed-up and 20x energy savings versus running the job on the mobile device, verifying the applicability of MCC in various scenarios.

1.2.2 Cloudlets

In the standard offloading scenario, the mobile device has a stable connection to the cloud to which it can offload computations and receive results back. However, the cloud need not necessarily reside on a remote Internet server. Satyanarayanan et al.[41] concentrate on using virtual machines to instantiate the required services on more powerful computing resources close to mobile devices, such as in a coffee shop. Such resources are called *cloudlets* and are formally defined as "a trusted, resource-rich computer or cluster of computers that's well-connected to the Internet and available for use by nearby mobile devices" [41]. Leveraging cloudlets has been shown to decrease the response time which is essential for many delay-intolerant mobile applications. The authors also describe that cloudlets facilitate serving multiple users in situations where bandwidth demand is near its peak. Customization of such cloudlets is less challenging than that of remote clouds. Note that stationary

cloudlets are complementary to our proposal of deploying MHPCs. Each of these approaches has its own advantages and disadvantages depending on the environment. Indeed, stationary cloudlets can be viewed as the same continuum of the MHPCs, where the difference is the time scale of (re)deployment. Another line of research related to stationary cloudlets involves a network design problem which attempts to place stationary HPCs to cover the computational needs of a group of users in an area. In [46], Shires et al. develop strategies for placement of tactical cloudlets, which are essentially parallel high performance computers (HPCs) in rough environments where the dominant infrastructure is ad hoc. They call this cloudlet seeding and define it as "the static strategic placement of HPC assets in deployed settings in such a way to balance computational load and limit hops to both stationary and mobile HPC nodes" [46]. Our problem is similar to this in that we must plan for computational needs of users in a geographic space, though because our HPCs are mobile, we have a route planning problem rather than a placement problem. The HPC seeding problem is also different to our proposal of combining the HPC and MFs as, again, the communication is provided through planning of the routes of the MFs and not the placement of the HPC(s). Finally, Zhang et al. focus on offloading algorithms considering the local load that users have and the availability of the cloudlets in an intermittently connected environment with mobile cloudlets by leveraging Markov Decision Processes (MDPs) [56].

1.2.3 Cyber Foraging

In [7], Balan et al. describe how MCC can opportunistically exploit the computing resources in a device's surrounding environment. In this architecture, known as cyber foraging, nearby computing and data storage servers are discovered during the course of a user's movement. These servers help improve the performance of the interactive applications and distributed file systems that are run locally. In an extended version

of this work, Flinn [19] introduces the concept of surrogate computing resources which are wired servers that possibly provide greater computing resources as compared to the mobile devices. Surrogates bear close similarity to the cloudlets introduced above. Taking the idea of using opportunistic resources one step further, Shi et al.[45] use intermittently connected clouds where the mobile device is assumed to have variable connectivity to the cloud. This, in essence, moves the computational dependence from a remote cloud to the mobile device's contacts which are opportunistically encountered. A second work of the same authors ([44]) explores improvements in the offloading of computation-intensive tasks of a mobile device to the cloud by both considering the performance seen by the user as well as the costs to the provider.

1.2.4 Vehicular Cloud Computing

Vehicular Cloud Computing (VCC) is formally defined as "[a] group of largely autonomous vehicles whose corporate computing, sensing, communication and physical resources can be coordinated and dynamically allocated to authorized users"[17]. In [52] Whaiduzzaman, et al. explain that VCC aims to exploit the underutilized computational resources of vehicles on streets, roadways and parking lots as computational nodes in a cloud computing architecture. Examples of where computational resources of vehicles can be used are cars parked in the long-term parking of an airport, a mall, or a small business. Another, more obvious example is utilizing the computational power of the large number of cars stuck in a traffic jam after a sporting event by the municipal authority, to reschedule traffic lights which will ultimately help dissipate the traffic jam as soon as possible [37]. VCC emphasizes the fact that while resources on these vehicles will be used for various services, there is no designated vehicle with the responsibility to provide services to other mobile or stationary users, thus issues of distributed cooperation and coordination are central[22]. While we also rely on

vehicles with computing or communication capacity, our work differs because we assume an MHPC or MF with controlled mobility that is given the task of supporting other users.

1.2.5 Data MULES

Another related line of research is the vast amount of work that proposes to utilize data MULEs (Mobile Ubiquitous LAN Extensions) [42]. By analogy to our MHPCs and MFs, MULEs are assumed to be equipped with a short-range wireless communication interface that can exchange data with nearby sensors that are encountered while moving. Similar to our MHPCs and MFs, MULEs can pick up data when in range of a source, buffer the data, and drop it off when they are in range of the destination. In contrast to our work, data MULEs usually follow a pre-determined path and initially broadcast their beacons so that stationary sensor nodes identify if they are on the MULE's path or can reach it through another node (initialization), and then in the subsequent rounds, data is collected from nodes on the MULE's path. These nodes pass their own data and the data of their children that cannot directly reach the MULE [26]. In contrast, our work mostly focuses on MHPCs or MFs with controlled mobility that are mostly responsible for helping nodes with computational tasks rather than data collection. The task scheduling component is also mostly absent in the data MULE line of work, while our work relies on properly scheduling the tasks on the MHPCs or on the HPC. We will discuss the value of this control over mobility in Chapter 4 of this thesis.

1.2.6 Message Ferries

Networking in challenging settings received research attention starting in 2005, under the term *disruption tolerant* or *opportunistic networks* [24]. Disruption tolerant routing could be used in combination with MHPCs or MFs to communicate tasks and results without requiring that the MHPCs and user nodes travel within radio

range. One piece of work in this area, with which we share our motivations partly, involves the original message ferrying literature. In a delay tolerant network, Message Ferries (MF) are a set of special mobile nodes that provide communication services for other (possibly stationary) nodes in the network [57]. Similar to our MHPCs, which by analogy can be called "Computational Ferries", Message Ferries enjoy controlled mobility in order to serve their purpose. Usage of Message Ferries is justified in crisis scenarios where infrastructure does not exist and nodes can easily go out of range of one another, in area sensing and surveillance applications where sensors are sparsely deployed, or in scenarios where economic or privacy considerations makes them a more suitable alternative [57]. Zhao et al. [58] suggest the possibility of using the stationary nodes or even multiple ferries as relays while exchanging data. In this framework, data might be picked up by one ferry, dropped at an intermediary node, and picked up and ultimately delivered by another ferry. Similarly, data can be exchanged among ferries before final delivery. Mukarram, et al. [10] present a variant of the Message Ferrying problem where limited mobility of nodes among which ferries move is allowed. This includes scenarios where nodes have a periodic mobility or a known mobility model. In contrast, our computational ferries provide processing services to nodes rather than data exchange services. In our frameworks, there is a possibility that the source and destination nodes are the same, but a task needs to be picked up, processed and returned. This incurs a necessary wait time in addition to the time needed to exchange data between the node and the ferry. Considering these examples, the emphasis of our work is on computation, not message delivery as considered in MF work. This makes our problem inherently different from that of the original MF. In our HPC+MF work, we use MFs as communication devices in their original form, but we add a processing unit to the picture which affects the requests for delivery of results.

1.2.7 Tactical Clouldlets

Another piece of literature that is of interest to our work is on Tactical Cloulets. This work is of interest of United States Army Research Laboratory [46, 50, 30, 12] and Department of Defense through the Software Engineering Institute (SEI) at Carnegie Melon University [35, 29]. This latter piece of work complements our work in the sense that it places an emphasis on developing a system that discovers the computation resources in the vicinity. In these works, specific attention is given to development of algorithms for code offloading, state synchronization and the deployment of tactical cloudlets is explored through development of the cyber-foraging architecture[7, 19]. This trend of work further verifies the feasibility of our proposal for deploying the MHPC and MF systems. Our work, in contrast to the above, focuses on the development of algorithms for controlling the mobility of the vehicles and for better scheduling of the tasks on the available compute resources and places less emphasis on the system architecture.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we present the problem of providing computation service to users in an area via high performance computers mounted on vehicles (MHPCs). In Section 2.1, we motivate the problem. Then we describe, in detail, the framework of the problem is Section 2.2. In Section 2.3, we investigate the problem from a systems point of view. We then formulate the problem in Section 2.4 and gaining some understanding of the fundamental features of it in Section 2.5. In Section 2.6 and 2.7, we use the lessons learned from the previous sections to propose scalable heuristics for scheduling the movement and task execution on the MHPCs, first with the assumption of complete knowledge of information into future, and then by lifting this assumption for a more general case. Finally, in Section 2.8, we perform extensive evaluations to observe the effect of various parameters on

the system.

In Chapter 3, we present the second problem, which suggests using Message Ferries (MFs) to provide communication for serving computation needs of user nodes via a stationary High-Performance Computer (HPC). In Section 3.1, we motivate the problem and describe its differences from the problem of Chapter 2. In Section 3.2, we present the framework of the problem. In Section 3.3, we look into the problem from a systems point of view. We then move into first formulating the problem in Section 3.4. In Section 3.5 and 3.6, we propose scalable heuristics for scheduling the movement and task execution on the problem, first with the assumption of complete knowledge of information into future, and then by lifting this assumption for a more general case. Finally, in Section 3.7, we perform extensive evaluations to observe the effect of various parameters on the system. Many sections of Chapter 3 can be compared side by side with those of Chapter 2 to gain a better understanding of the systems and their differences.

In Chapter 4, we introduce a new dimension to the problem: controlling the mobility of the vehicles. In Section 4.1, we present a new classification of the problems solved in Chapter 2 and 3. In Section 4.2, we present a counterpart for the heuristics of Chapter 2 that does not control the mobility of the MHPCs. Similarly, in Section 4.3, we present a counterpart for the heuristics of Chapter 3 that does not control the mobility of the MFs. Finally in Section 4.4, we use these newly developed heuristics to perform more experiments that reveal the value of controlling mobility of the vehicles in Chapters 2 and 3.

In Chapter 5, we introduce a heuristic for obtaining mobility traces from contact traces. In Section 5.1, we introduce integrating unconditional mobility of user nodes as the next logical step for the work of Chapter 2 and 3, and we motivate inference of mobility traces from contact traces as a more technically feasible method and describe how obtaining of mobility traces can help integration of the user node mobility into

the problems of Chapter 2 and 3. In Section 5.2, we formally introduce our heuristics for inferring mobility traces from contact traces. Finally in Section 5.3, we evaluate the proposed heuristics and compare it to previous work.

CHAPTER II

COMPUTATIONAL FERRIES: SCHEDULING FOR MOBILE HIGH PERFORMANCE COMPUTING

2.1 Introduction

In this chapter, we introduce the first of the problems proposed in Section 1.1. In this problem, we consider the case where computationally intensive computations are off-loaded to a more powerful computing resource mounted on a vehicle. We refer to this vehicle as a Mobile High Performance Computers(MHPCs). These computations are usually intensive tasks owned by users[1] dispersed in an area with no Internet connectivity infrastructure.

Our presentation of this problem, named the MHPC problem, focuses on the controlling of the mobility of this computing resources (MHPCs) as well as proper scheduling of computation on the processors of these vehicles in order to achieve various objectives including finishing computation as soon as possible, minimizing travel and fuel consumption, providing the most timely service to all users in the area with computationally intensive tasks, etc. Complementing our work is the Tactical Cloudlet work from the Software Engineering Institute (SEI) [29] that focuses on development of a detailed system architecture and communication protocols that allows the users to utilize the nearby computational resources. While that work provides a framework for the components of problems similar to ours to interact with each other in the field, our work focuses more on the abstract features of the problem and proposing best solutions, in lieu of a minimalistic solution, for serving the user nodes

[1] Interchangeably called "user nodes" in this text.

As mentioned in Chapter 1, this problem finds its applications in the situations where connectivity to the infrastructure Internet is not readily available and also when the area of coverage is large relative to the reach of communication channels that can support a reasonable bandwidth for communication of the tasks. Notably, in the former case, traditional cloud computing solutions [16, 18] can address the problem, while in the latter, placement of a stationary cloudlet that can be reached by all users in its vicinity [46, 12, 56, 30] seems to be a more viable solution. Hence, using mobility-controlled vehicles (MHPCs) to serve the computational tasks of all users in the area is useful in an environment without traditional communication infrastructure and large enough that cloudlet seeding[46] is not a feasible solution.

To illustrate the problem setting, consider the simple system shown in Figure 2. An MHPC travels in the region at an average speed of 10 m/s. It has a single preemptive processor[2]. Assuming that we describe distances as driving time of the MHPC, user Node 1 is located five minutes away from the current location of the MHPC, while User Node two is located two minutes away. The user nodes are four minutes apart from each other. In this simple example, the user nodes are stationary for the time period under consideration. Each user node generates a task at time 0, Node 1 has a task that will take 10 minutes on the MHPC, while Node 2 has a task that will take three minutes. Assume that the processor on the MHPC allows preemption, thus a task that is started on the processor can be interrupted to allow work on another task, with resumption later from the point of interruption. The MHPC can compute and travel simultaneously.

In what order should the MHPC visit the task locations? In what order should the processor work on the two tasks? What is a reasonable metric for quality of a solution? What changes if task 2 is not known to the MHPC until two minutes into the scenario? Our MHPC framework and solutions address solutions to such

[2]For a formal definition of preemptive processing, refer to Section 2.2.4

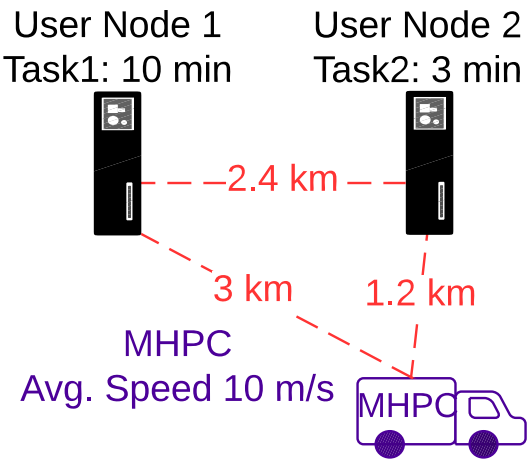


Figure 2: A simple example of MHPC scheduling problem with two nodes and a single processor MHPC. Sample task and MHPC features and distances among entities are shown on the figure.

problems.

We describe a formal framework for the above mentioned problem is Section 2.2. We then propose a mathematical formulation of this problem is Section 2.4 and use this formulation to derive some insights and develop scalable heuristics for a version of the problem that assumes complete in-advance knowledge in Section 2.6. In Section 2.7, we extend the previous heuristics to deal with more realistic scenarios where information is revealed as users request computational assistance from the MHPC. Finally, we present evaluations in Section 2.8 that further our knowledge of how the system is affected by various parameters and how to make decisions in choosing resources when serving a given scenario.

2.2 Problem Framework

2.2.1 Framework Structure and Problem Settings

In this section, we describe the general framework in which the MHPC problem is studied. We consider a system with mobile user nodes traveling in a bounded geographic area. Also present in the area are V MHPCs, initially located at known locations. Each MHPC is assumed to have m processors; i.e. it is capable of executing

m computational tasks simultaneously, one per processor. The vehicles on which MHPCs are mounted are assumed to be identical and to take a known constant time to travel between a pair of nodes as a function of locations of those two nodes. Finally an MHPC Controller is present in the area that coordinates the efforts of the MHPCs to serve the tasks. This will be discussed in more detail in Section 2.3. We introduce the problem with these simplifying assumptions to streamline the exposition and allow the key insights to be highlighted. We challenge and relax some of these assumptions later in the thesis.

We assume that user nodes notify the MHPC Controller of the existence of computational tasks for offloading, using a long-range, low-bandwidth radio such as the ones suggested in [57, 9]. This radio is assumed to be only useful for control plane operations and not capable of exchanging tasks/results. The user nodes are mobile, thus to facilitate task and result exchanges in the future, it is necessary to provide for meetup opportunities. In our formulation, the user node specifies a location where the MHPC can pick up the task and a location (possibly different) where the MHPC can deliver the result. Without loss of generality, we demonstrate two examples of specifications of such locations:

- In a military setting, these locations can be considered as stationary Forward Operating Bases (FOBs) used as readily available resources for tactical missions without the need of reacquiring or moving resources. These FOBs may have inexpensive storage capabilities similar to throwboxes in Delay Tolerant Networks [59], and while soldiers can move, they can place their tasks in their corresponding FOB after notifying the MHPC and receive the results from the same or a different FOB. An example of this setting is shown in Figure 3.

- In a more general setting without FOBs as meeting locations, we assume there are a finite number of locations designated exclusively to facilitate task pickup

and delivery. These could be chosen, for example, based on prior semantic information about the geographic field, such as safe spaces, common areas for mobile node congregation, etc. The algorithms will also work if the meeting locations are arbitrary (not chosen from a finite set of fixed locations), but fixed locations enhance the scalability.

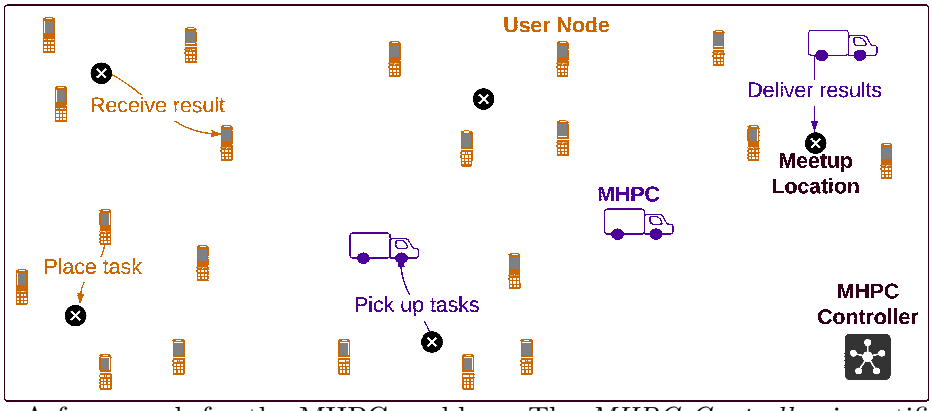


Figure 3: A framework for the MHPC problem. The *MHPC Controller* is notified about the tasks at the *User Nodes* which are served by the *MHPCs*.

As far as the MHPCs and MHPC Controller are concerned, tasks arrive in groups of one or more generated at different locations. In a general case, this can imply some batching of the tasks before they are processed by the MHPC. As we will describe in Section 2.7, it may save some re-calculations of the solution if arriving tasks are revealed in groups rather than individually. Each task, i, in a group is represented by the vector $(R_i, D_i, T_i, LP_i, LD_i)$, where R_i is the time that task i is *available* for pick up at the location, LP_i. The value of R_i denotes the time taken for the user node to get to the designated location. D_i is the deadline for the task, after which the user node is no longer interested in the result of the processing, and LD_i is the designated delivery location for the results of the processed task. In the case that $LP_i = LD_i \ \forall i$, we call them *task locations* ;otherwise we call them pick up locations or delivery locations based on designation. Finally, T_i is the duration of time that it takes to process the task on one of the processors of an MHPC.

After being notified of the task, the MHPC Controller can commit an MHPC to the task, in which case it will schedule the pick up, computation and delivery of results to complete before the deadline, or it can reject the task. One interpretation of the deadline is as an estimate of the local running time of the task. In this case, the rejection of the task by the MHPC means that the user node will obtain the result faster if it processes it locally. Further work on offloading decisions can be found in [13]. Another interpretation of a deadline is an estimate of the remaining power on the user node's hand-held device. In this case, delivery of the result will not be helpful, since the device's battery will be depleted before the result is obtained.

Note that there is significant flexibility in our task arrival and availability formulation. For example, we can model the scenario where all tasks are immediately ready by setting all availabilities of tasks to the corresponding times when they are generated. Conversely, we can relax the deadlines by setting them to a sufficiently large value.

An MHPC can *pick up* a task if it visits the pick up location of the task at or after the task availability time. The MHPC travels in the region, picking up available tasks, scheduling and completing their execution on its processors, and delivering the results. As it will be described in Section 2.7, the arrival of groups of tasks affects the MHPCs' schedule for new pickups, deliveries, and processing, but does not affect its previous commitments. Note that there is no recommendation about picking up all available tasks first and then delivering all the results, or serving one task completely before getting to another. All such decisions are made only to achieve the minimal objective value.

Pick up and delivery of the tasks are either done through a wired connection, in case of user nodes exchanging tasks through FOBs or using a short-range, high-bandwidth radio in the case of a designated location. We assume that the transmission range of this radio is negligible compared to the dimensions of the field where MHPCs

travel; otherwise, instead of using MHPCs, a stationary HPC can be placed in a proper location where it can communicate with all user nodes. We assume that any time required for task pick up and result delivery is negligible compared to travel and execution time. This assumption can also be lifted by adding a communication delay proportional to the size of the task / result for each data exchange.

We also assume that the MHPC schedule is computed in a bounded amount of time by the MHPC Controller, given the current schedule of all MHPCs and the newly obtained groups of tasks. This time is assumed to be negligible compared to the duration of the tasks. Finally, we assume that each MHPC can store all received tasks pending their execution and can store all execution results pending delivery; i.e. the size of the buffers at the MHPCs is much larger than the amount of information that they require to hold at a time.

An important feature of our model is that the MHPCs can *compute while moving* so that travel times between nodes can be used to work on tasks that available at the MHPC. This implies that if a better solution requires so, the MHPC can pick up a task and start processing it while moving towards another pickup or delivery.

2.2.2 Structure of the Solution

We are interested in producing a schedule for our MHPC system that has three components:

- **MHPC Assignment:** This specifies the subset of tasks that are assigned to each MHPC. Each MHPC will be responsible for pick up, processing and delivery of its assigned tasks. No inter-MHPC communication is used during job processing. The tasks that are not assigned to any MHPC will be rejected since they cannot be delivered by the desired deadline.

- **MHPC Tour:** This is a pick up / delivery location visit schedule for each MHPC and comprises a vector of tuples in the form (*location id*, *location visit*

time, *location visit duration*). The location visit time specifies the time that the MHPC arrives for pick up / delivery. For example, an entry $(\pm i, r_{\pm i}, w_i)$ specifies that the MHPC arrives at location i for pick up $(+)$ or delivery $(-)$ at time $r_{\pm i}$ and waits for an amount of time w_i to be able to perform the pickup/delivery[3].The location visit duration specifies how long the MHPC pauses at a location during a visit. Unlike the standard traveling salesman problem, where this time is always 0, the MHPC may find it advantageous to arrive early and wait either for task completion or for a task to become available for pick up.

- **Task Execution Schedule:** This is the schedule of task execution on each processor for each MHPC. This component comprises m vectors, one for each of the m available processors of the MHPC. Each vector consists of tuples in the form (*interval*, *scheduled task*), where *interval* denotes the time period when the MHPC is working on the specified *scheduled task*. For example, an entry $((t_1, t_2), i)$ for processor j specifies that processor j is working on task i from time t_1 to t_2, inclusive.

2.2.3 Objective Value

In this section, we describe the various objective values that can be minimized as our goal in the MHPC problem. Note that in our implementation, objective value of the problem is an independent module that can be replaced based on the specific requirements of the instance of the problem to be solved. Any objective value that depends on travel times and task durations and other known inputs of the problem can be interchanged regardless of the details of the algorithms used to solve the problem.

A few possibilities for the objective value that we have considered are described

[3]Such wait occurs due to the MHPC arriving at the pick up location earlier than the task is available or the MHPC arriving at the delivery location before the task has completed its processing

below. These objective values can be either directly integrated into the instance of the problem that is being solved or they can be calculated on a solution that has been provided for the problem to study the effect of minimizing one objective value on the value of another. For example, we study the effects of providing a timely service on the distances traveled by the MHPCs.

- **Return Time (RT):** This is one of the simplest and most natural objective values for the problem. RT is defined as the time that it takes for the latest MHPC to finish processing of its last task and deliver it to the designated location. This is used if total time, in the view of the MHPC, is of most interest.

- Time Throughput (TT): TT is similar to RT. It is defined as n/RT where n is the total number of tasks that the system has processed up until the current time. In essence, TT represents the number of tasks served per unit time that the system was running. One must note that TT shall be maximized for better performance as a larger value means that more tasks are served in the same amount of time.

- **Average Completion (AC):** AC is defined as the average of the delivery moments for all tasks by all MHPCs. This is useful, if providing early service to all user nodes is of most interest.

- **Average Flow (AF):** AF is the average of the difference between pick up and delivery for all tasks served by all MHPCs. AF modifies AC by penalizing lateness in the delivery of the result only relative to the time that the task is revealed and avoids skewing to penalize tasks that are revealed later more than the earlier tasks.

- **Average Wait (AW):** AW is the average time that MHPCs wait in delivery

locations before they can deliver tasks. This can be useful if vehicle needs to have high mobility due to strategic reasons.

- **Average Travel (AT):** AT is the average time that each MHPC is moving. This metric might be of interest in the cases that fuel consumption of MHPC is an important metric to be minimized.

- **Distance Throughput (DT)**: DT is similar to AT. It is defined as $1/AT$ or total number of tasks that the system has processed up until the current time divided by the total distance traveled by all MHPCs. In essence, DT represents the number of tasks served per unit distance that the MHPCs have traveled. One must note that DT shall be maximized for better performance as a larger value means that more tasks are served while traveling the same.

- **Normalized Average Flow(NAF):** NAF is defined below:

$$NAF = \max_{k \in Vehicles} \left(\underset{i \in Tasks}{\text{Average}} \frac{r_{-i} - R_i}{T_i} \right)$$

where, r_{-i} indicates the time when the results of the task are delivered. Other notations were introduced earlier in this section. This metric, like AF, envisions providing a timely service to each task that is requested relative to the time that the task is revealed, but it also normalizes this value by task duration so that larger tasks are not penalized for worse service simply due to their duration being longer than shorter tasks. Note that this metric will be equal to one if there is a dedicated MHPC that is present at the location of each task before it is revealed.

2.2.4 Work Conservation, Preemption, and Processor Sharing

In this subsection, we describe the options for the scheduling of task execution on MHPC processors.

Work Conservation: The MHPC may deploy either *work conserving* or *non-work-conserving* scheduling. In a work conserving schedule the MHPC never queues a task if there are available processors. A non-work-conserving schedule, on the other hand, may allow a task to be queued (not immediately processed) even though there are available idle processors. We will show later than work conservation is implied in the choice of preemptive or non-preemptive scheduling (defined below) in our model.

Preemption: The MHPC may allow preemption or not. In a schedule that allows preemption (preemptive schedule), a task already in process may be paused in favor of another task. The preempted task is resumed at a later time. In general, there are two types of preemption, *non-resume* and *resume*. In the former, when a task is preempted, it must start from the beginning when scheduled again, while in the latter the task can start from where it was left off. We use *preemptive resume* in this thesis. A non-preemptive schedule on the other hand requires a task that is scheduled to completely finish processing before another task can start processing on the same processor. Our model allows both preemptive and non-preemptive scheduling. We assume that all these processors in the MHPC are of the same compute power.

Processor Sharing: This is also a possibility for task scheduling. With processor sharing, the same processor can process k tasks in parallel with $1/k$ of its processing power. Our model, however, for task processing on the MHPC's m processors assumes *no* processor sharing. That is each processor is allocated at most one task at a time.

2.3 System Architecture

In his section, we describe the components of the system depicted in Figure 3 from a system's standpoint. The details of this section can help future researchers to implement their own version of our system.

A system deploying MHPCs shares many of the same challenges faced by opportunistic communication systems. These challenges include concerns for accommodating various user node mobility patterns as well as using mechanisms for neighbor discovery that allow for efficient use of communication opportunities. In addition, an MHPC system, similarly to systems using message ferries [10, 57], requires careful scheduling of ferry mobility as well as efficient coordination among the ferries when more than one is used [58]. A computational ferrying system, however, possesses significant additional features that warrant special attention. These stem primarily from the fact that the MHPC is not simply delivering data but also performing computation. While data transmission is required to and from the MHPC in order to perform the computation offloading, there is an additional requirement to deliver the result when computation is completed. Furthermore, an MHPC is *performing computation while moving*. These features require new techniques for scheduling of movement and computation on an MHPC. We develop an architecture for such an MHPC system in this section.

Figure 4 provides a high-level overview of the main components of the MHPC system. It consists of three components: a number of *User Nodes* that generate computational tasks and seek service from MHPCs; a set of *MHPCs* which are vehicles with a high performance computer on-board that are responsible for providing the actual service; and an *MHPC Controller* that manages the requests received, and dictates to MHPCs their future plan for picking up, processing and delivering tasks. We also describe the *Radio Environment* through which the communications happen as a part of this architecture.

Radio Environment The three components can exchange high-level task metadata using a long-range, low-bandwidth radio such as those suggested in [57, 9]. Wide-coverage, low-bandwidth infrastructure using unlicensed bands may also be

available and has been deployed [47]. This radio is assumed to be only useful for control plane operations and not capable of exchanging tasks/results. Actual tasks and results are exchanged via a short-range, high-bandwidth channel. Lewis, et. al [29] investigate a detailed client-server architecture along with the proper communication protocols for such exchanges of information.

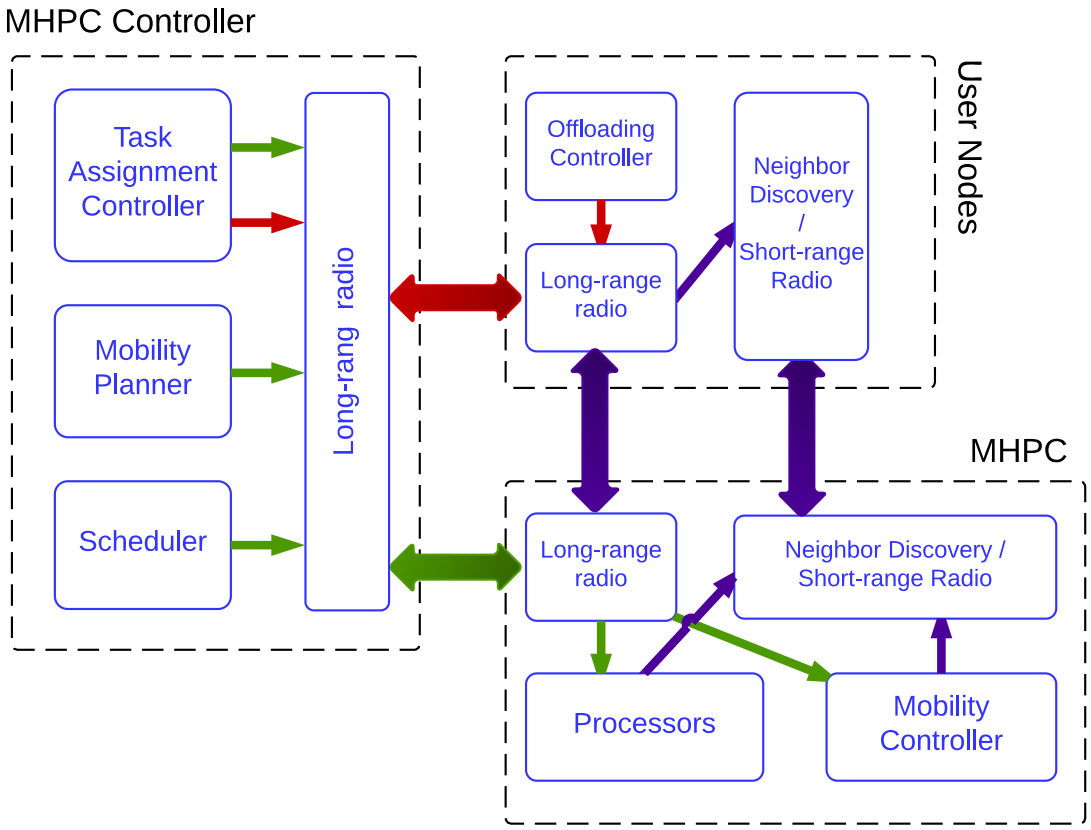


Figure 4: A system architecture for the MHPC problem. Three components of the system are: (1) *User Nodes* that own the tasks, (2) *MHPCs* that provide the communication and computation service by picking up, processing, and delivering the tasks, and (3) *MHPC Controller* that plans the mobility and task execution schedule on the MHPCs.

User Nodes User Nodes are the owners of the tasks. Once a task is generated at a User Node, its **Offloading Controller** module decides whether the request must be sent with the task metadata to the MHPC Controller or the tasks must be processed locally. Various reasons ranging from privacy to time trade-offs may contribute to a decision to process the task locally, the details of which are outside the scope of

this work. The interested reader can refer to [45] for examples of work focusing on offloading decisions. Once the User Node sends out the task metadata, if it does not receive a rejection via the **Long-range Radio** from the controller, it will continue communicating with the assigned MHPC if necessary until the MHPC is within its range so that the task/result can be exchanged via the **Short-range Radio** module.

MHPC The MHPCs are the providers of computational service in the system. Multiple MHPCs can operate in the same area. In addition, each MHPC can have multiple processors and is able to execute several computational tasks simultaneously. The MHPCs receive their assigned tasks, mobility plan and task execution schedule from the controller via the long-range radio. The **Processors** of the MHPC work on the picked up tasks according to the specified execution schedule. They also report on the state of their allocated tasks to the Controller over the same radio. The **Mobility Controller** follows the MHPC tour indicated by the controller to visit the User Nodes for pick up and delivery. If the User Nodes are mobile, this module needs to use the long-range radio to repeatedly query the User Nodes for their current location and plan accordingly to meet them. Once the MHPC is in range with a User Node as notified by its **Neighbor Discovery** module, it can exchange the corresponding tasks/results to the User Nodes.

MHPC Controller This component handles the task offloading requests received from User Nodes. These requests are in the form of task metadata using a long-range and low-bandwidth radio. The MHPC controller uses knowledge of MHPC and User Node locations as well their states to compute a *computation and mobility schedule* for the MHPCs the components of which are detailed in Section 2.6 and 2.7. The computation and mobility schedule is communicated to the MHPCs and updated as needed via the long-range radio.

In this thesis, our main focus is to propose mechanisms for MHPC Controller to

provide the best computation and mobility schedule.

2.4 Mathematical Model of MHPC Problem

In this section, we propose a mathematical model to describe the MHPC problem of Section 2.2. To model the MHPC problem, we need to make further simplifying assumptions to the general problem stated in Section 2.2. We consider complete knowledge of the tasks into the future. We also restrict this formulation to have a single MHPC with a single non-preemptive processor serving tasks $1, \ldots, n$. Further, we assume that all tasks are available to be processed from time zero and there is no deadline[4].

Tables 1a and 1b summarize the parameters and decision variables used in the following formulation of the MHPC problem for the above framework.

Table 1: Parameters and Decision Variables used in the MHPC formulation.

(a) Summary of known parameters in the problem formulation.

Param.	Interpretation
$+i$	Id for pick up location of task i
$-i$	Id for delivery location fo task i
T_i	Time duration of task i
t_{ij}	Travel time between location of tasks i and j
M	A very large number
$\pm\mathcal{N}$	Set of all pick up and delivery locations

(b) Summary decision variables in the problem formulation.

Decision Var.	Interpretation
x_{ij}	1, if task at location j is visited right after i and 0 otherwise
$r_{\pm i}$	Time that task at node i is picked up/delivered
w_i	Waiting time at location of task i before delivery
s_i	Time that task at location i starts processing
y_{ij}	1, if the task at location j is scheduled right after i and 0 otherwise

In this model, we represent the single MHPC as node 0 and the task at location

[4]We have formulations for multiple processors, multiple vehicles, non-zero availability, and non-zero deadlines which are skipped in this proposal due to space constraints. In Section 3.4, we present a slightly more complicated version of a similar problem

1 through n as $1, \ldots, n$[5]. For simplicity, we formulate the problem such that the MHPC returns to its starting location. First, we makes two copies of each of these tasks. We label one copy as $+0, +1, \ldots, +(n-1)$ to indicate pick up locations, and label the other copy as $-0, -1, \ldots, -(n-1)$ to indicate the delivery locations[6]. In this notation, ± 0 refers to the starting location of the MHPC starts. This reduces the problem to a combination of a Traveling Salesman Problem (TSP) that finds the minimum cost tour of $\{+0, +1, \ldots, +(n-1), -0, -1, \ldots, -(n-1)\}$ given some precedence and scheduling constraints and a Processor Scheduling Problem. The objective value of this formulation is RT (refer to Section 2.2.3 for definition.)

These constraints can be described as follows:

- A tour must visit all of the locations $\pm i$ in some order. This means that all the tasks shall be picked up, processed and delivered before completing the tour.
- Delivery location $-i$ can only be visited after pick up location $+i$ is visited. This means that a task can only be delivered if it has already been picked up.
- Processing of task i must start sometime after pick up location $+i$ is visited. This means that processing of a task must start sometime after its pickup.
- If the task i is not finished by the time that delivery location $-i$ is visited, we shall wait until the task is finished. This means that we may need to wait at delivery locations so that the completed task can be returned to the node.
- If task j is scheduled right after task i and all the above constraints are satisfied in such scheduling, it must start at a time later than the time task i has started processing plus the duration of the task, T_i. Note that task j starts either

[5]There is no requirement that locations $1, \ldots, n$ are distinct, hence multiple tasks can be generated at the same location.

[6]In general, the pick up and delivery location can be distinct. In this formulation, we assume they are the same for each task. Extension to distinct locations is straightforward.

immediately after task i if it has already been picked up, or it starts once the task has been picked up.

This problem can be formulated as a Mixed-integer Linear program (MILP) as follows:

$$\underset{x,r,w,s,y}{\text{minimize}} \quad (r_{-0}) \tag{1}$$

$$\text{subject to} \quad \sum_{j\in\mathcal{V}} x_{ji} = 1 \quad ;\forall i \in \mathcal{V} \tag{2}$$

$$\sum_{j\in\mathcal{V}} x_{ij} = 1 \quad ;\forall i \in \mathcal{V} \tag{3}$$

$$x_{-0,+0} = 1 \tag{4}$$

$$s_i \geq r_{+i} \quad ;\forall i \in \mathcal{N} \tag{5}$$

$$s_i + T_i \leq r_{-i} + w_i \quad ;\forall i \in \mathcal{N} \tag{6}$$

$$r_{-i} \geq r_{+i} \quad ;\forall i \in \mathcal{N} \tag{7}$$

$$r_{+i} + t_{ij} - (1 - x_{ij})M \leq r_j \quad ;\forall i \in \mathcal{N}\ ,\ \forall j \in \mathcal{V}\ ,\ j \neq i, 0 \tag{8}$$

$$r_{-i} + w_i + t_{ij} - (1 - x_{ij})M \leq r_j \quad ;\forall i \in \mathcal{N}\ ,\ \forall j \in \mathcal{V}\ ,\ j \neq i, 0 \tag{9}$$

$$r_{+0} = 0; \quad s_0 = 0; \quad w_0 = 0 \tag{10}$$

$$s_i + T_i - (1 - y_{ij})M \leq s_j \quad ;\forall i \in \mathcal{N}\ ,\ \forall j \in \mathcal{N}\ ,\ j \neq i \tag{11}$$

$$y_{ij} + y_{ji} = 1 \quad ;\forall i \in \mathcal{N}\ ,\ \forall j \in \mathcal{N}\ ,\ j \neq i \tag{12}$$

$$x_{ij} \in \{0, 1\} \quad ;\forall i, j \in \mathcal{V}\ ,\ j \neq i \tag{13}$$

$$y_{ij} \in \{0, 1\} \quad ;\forall i, j \in \mathcal{N}\ ,\ j \neq i \tag{14}$$

$$r_i, s_i, w_i \geq 0 : \text{Free} \quad ;\forall i \tag{15}$$

In the above formulation:

- Objective (1) minimizes r_{-0}, which is the time that the moving vehicle returns to the initial location. This is the Return Time, RT, described in Section 2.2.3.

Since the formulation forces -0 to be the last stop in the tour, r_{-0} is the time that all tasks have been pickup, processed, delivered, and the vehicle returns to its starting location. This implies that we want this to finish the service as soon as possible.

- Constraints (2) and (3) confirms that each pick up / delivery location in $\mathcal{V}$ is entered once and exited once respectively, i.e. visited one time and no more.

- Constraint (4) ensures that the tour starts at $+0$, initial location of the MHPC, and ends at -0, the final location of the MHPC.

- Constraints (5) and (6) enforce that processing of task i starts after the task has been picked up and ends before the time that the delivery location is to be visited plus the waiting time at the delivery location.

- Constraint (7) indicates that task i needs to be pickup before it is delivered.

- Constraints (8) and (9) indicate that visit time of the consecutive pick up or delivery locations shall be separated by, at least, the corresponding edge traversal time (plus the waiting time at a delivery location) provided that first location is visited for a pickup (delivery) location.

- Constraint (11) indicates that if task i is processed before task j, the starting time of the processing of these tasks shall be separated by at least the duration of the task, T_i.

- Constraint (12) indicates that either task i is processed before task j or vice versa.

- Constraints (15) indicate that the values of visit times, start of processing times and wait times can be any positive real numbers.

- Constraints (13) and (14) indicate that the variables x and y are binary.

Solving the MILP Problem: A typical solver will attempt to solve this problem using a branch-and-cut or branch-and-bound method. The former procedure manages a search tree consisting of nodes. Every node represents a linear subproblem (LP) to be solved and checked for integrability. A branch is the creation of two new nodes from a parent node and occurs when the bounds on a single variable are modified. In our case, for binary variables, two new nodes, one node with a modified upper bound of 0 (the downward branch), and the other node with a modified lower bound of 1 (the upward branch) are created which will yield distinct solution domains. A cut is a constraint added to the model. The purpose of adding any cut is to limit the size of the solution domain while not eliminating legal integer solutions. It can be shown that such a method smartly reduces the number of possibilities to be checked on binary variables compared to the brute-forced method of checking all possibilities. But in the case of TSP, and hence or problem, it cannot reduce the order of such possibilities to be checked beyond a factorial order, which makes the problem to remain NP-hard.

2.4.1 Complexity Analysis for an Exact Algorithm

The MHPC problem is easily seen to be NP-Hard. A special case of the MHPC problem occurs when there is a single vehicle with a single processor and all the tasks are known in advance (A). A special case of (A) is TSP when the the minimum travel time among every location is larger than the maximum duration of the processing of the task. Since TSP is NP-Hard, (A) and MHPC are as well. It can be shown that even the simplified version of the MHPC problem presented above has $\frac{(2N)!(N!)}{2^N}$ feasible solutions. As a result, a brute force approach of trying all possibilities to find the solution is ineffective even for small number of nodes. This drives us to study the formulation and understand basic features of the problem that allow us to propose scalable heuristics for the problem.

2.5 Theoretical Foundations

The following lemmas capture some understanding gained by studying the concepts of work-conservation and preemption introduced in Section 2.2.4 and the formulation of Section 2.4.

Lemma 1: If task preemption is allowed in the MHPC scheduling then an optimal schedule is always work conserving. In a work conserving schedule the MHPC never queues a task if there are available processors.

Proof: To see this, if we discretize the time with the pick up and delivery moments, in any non-work-conserving solution, there exists at least one node i for which its task is picked up at time r_i but the CPU has been kept idle until the next pick up at some $r_j > r_i$. Since the preemption is permitted, the non-usage of the interval $[r_,r_j]$ can result in a longer waiting time at the delivery of i or a node picked up before i and delivered after it, which could have been avoided (alleviated) if i or the other node would have been processed in (a part of) this interval.

Lemma 2: An optimal schedule for the preemptive MHPC with resume given a specific visit order, is one that visits the accepted tasks of the MHPC in the order of delivery. Preemption might happen when a newly picked up task is scheduled to be delivered earlier.

Proof: It suffices to propose an example where an non-work-conserving is optimal. Such setting is depicted in Figure 5 for a single processor MHPC . Assume that the job at user node 1 is much longer than the other two, i.e., $T_1 \gg T$, also assume the processing time of this job is at least twice its distance from user node 2, i.e., $T_1 > 2t$. Finally assume that depot location ,0, is much closer to user node 1 than 3, i.e., $t'_0 \gg t_0$. It can be shown that an optimal visit order is as follows: $[+0 \rightarrow +1 \rightarrow +2 \rightarrow -2 \rightarrow +3 \rightarrow -3 \rightarrow -1 \rightarrow -0]$ the optimal schedule holds processing task at user node 1 once picked up and only starts this task, after the task at user node 3 is picked up and finished processing. Such solution is non-work-conserving and optimal

with a total time of $t_0 + t + T + t + T + T_1 + t_0$, any work-conserving solution that processes task at user node 1 immediately after pick up can be shown to have a total time of $t_0 + T_1 + T + t + T + 2t + t_0$ which is t time units longer. This means that the work-conserving is not always optimal for the MHPC Scheduling Problem with non-preemptive scheduler.

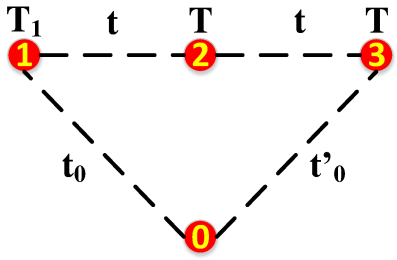


Figure 5: Example used in the proof of Lemma 2. A single processor MHPC is initially placed at location 0. There are jobs of durations T_1 at locations 1 and jobs of duration T at locations 2 and 3. Time taken to travel between any two locations is shown in the figure.

In the algorithms presented in Section 2.6, we use these lemmas to determine the order of processing the tasks on each MHPC.

2.5.1 Bounds on the Performance of the Offline Heuristic

In this section, we propose two simple upper and lower bounds on the optimal solution of the formulation in this section. These bounds are derived by decoupling the tour component from the task scheduling component.

When the problem sizes are too large to solve the optimization formulation, these bounds can be sued to compare the performance of the heuristic to these bounds.

- **Upper Bound:** A trivial but non-efficient solution to the problem is a TSP tour of the tasks $0, 1, \ldots, n-1$ that visits each task location, picks up the all tasks at that location, stays in place and processes them, delivers the result, and then visits the next task location on the tour.

  The total completion time under such scenario is $t_{TSP^*} + \sum_{i=1}^{n} \sum_{k=1}^{l_i} T_{ik}$, where

t_{TSP^*} is the time for the optimal TSP tour. Note that finding t_{TSP^*} is an NP-hard task. An efficiently computable (though of course looser) upper bound can be found by replacing the optimal time with the result from well-known polynomial approximation algorithms for TSP. It is obvious that any optimal solution to the problem will result in a completion time at most equal to this bound.

- **Lower Bound:** A simple lower bound ignores the requirement to travel between task locations and instead is the completion time of the best schedule for processing of all tasks. A lower bound on this time is given by $max(\sum_{i=1}^{n} \sum_{k=1}^{l_i} T_{ik}/m \ , \ \underset{i}{max}(T_{ik}))$ for a non-preemptive scheduler and $\sum_{i=1}^{n} \sum_{k=1}^{l_i} T_{ik}/m$ for a preemptive scheduler.

2.6 Offline MHPC Problem

In this section, we turn into development of heuristics that consider the offline version of the problem. Since the MHPC problem, as described in Section 2.4, is NP-Hard, we need to develop efficient algorithms to solve it. On the other hand, it is obvious that at the heart of the general problem described in Section 2.2 is an assignment of tasks to MHPCs given their commitments (similar to a bin-packing problem), a pick up / delivery order planning (similar to TSP) and a task scheduling problem that is solved on the processors of each MHPC. This encompasses the offline problem studied in this section. Since greedy algorithms are widely used to solve each of these problems, we adopt a similar approach to solve the offline MHPC problem.

To offer a scalable solution, first, we propose a greedy heuristic that builds a solution by considering one task at a time and assigning it to the best vehicle in the best tour location and task execution schedule assuming complete knowledge of tasks. In the next chapter, we extend this to non-complete knowledge. We provide one construction heuristic and one improvement heuristic and describe the applicability

of each below.

2.6.1 Constructive Heuristic

The constructive heuristic builds the solution from scratch inserting tasks one by one until all of them are placed. It uses a greedy approach to do so.

We describe a constructive heuristic that accounts for both preemption and non-preemption. The constructive heuristic starts with an empty MHPC Tour and Task Execution Schedule (as described in Section 2.2.2) and then considers the tasks at user nodes one at a time and determines the best place in the current tour of some vehicle. To select which vehicle and tour location (or equivalently task) to visit at each step, the heuristic tries the current task under examination in each possible position in the tour of each vehicle for pick up and delivery. For each of these placements that does not violate the precedence and deadline requirements, it finds the best schedule. If scheduler is preemptive, this schedule is earliest delivery order first according to the visit order that the tour forces. If the scheduler is non-preemptive, all $n_k + m$ positions for a candidate processing order on an MHPC with m processors with the addition of the new node into a tour of currently n_k tasks are tested and the one that gives minimal increase in the objective value is chosen. At the end of each iteration, the MHPC Assignment, MHPC Tour and Task Execution Schedule that gives the smallest increase in the objective value is chosen. The offline heuristic for preemptive case is formally described in Algorithm 1.

The order in which tasks are visited for this greedy heuristic, noted in Algorithm 1, can be chosen in various ways including choosing tasks by index order, choosing the nearest (farthest) node to the MHPC, choosing the nearest (farthest) task to the latest currently inserted task. Alternatively, a "pure" greedy approach can be used that considers every task at every step and chooses the task that gives minimal increase in the objective value for final placement. This approach requires considerably more

Algorithm 1: Offline MHPC heuristic pseudo-code

Input: Number of tasks: n, Specifications of each task i: $(R_i, D_i, T_i, LP_i, LD_i)$, Number of MHPCs/processors per MHPC: (V, m)

Output: *Solution* = MHPC Assignment (per instance): $tasks_k \; \forall k \in V$, MHPC Tour (per MHPC): $tour_k = \{\ldots, (\pm i, r_{\pm i}, w_i), \ldots\}; \forall k \in V$, Task Execution Schedule (per MHPC): $sched_k = \{((t_1, t_2), i)\} \; \forall k \in V$

```
Initialization: Mark all tasks as not-visited;
Sort the tasks to examine in some order;
while There exists not-visited tasks do
    Take the next not-visited task, i.
    for Each MHPC, K, in V do
        for Each placement of task i's pick up and delivery in the current
        MHPC tour that does not violate the deadline and precedence
        requirements do
            Schedule task execution according to an earliest delivery order first;
            Store this assignment, tour, and execution schedule as Temporary
            Solution.
            if Temporary Solution Increase the objective value minimally then
                Choose Temporary Solution as Solution.
            end
        end
    end
end
Based on the Task Execution Schedule and MHPC Tour in Solution, calculate
r_{±i} and w_i for each pick up / delivery. return Solution, Non-scheduled tasks as
rejected.
```

computation time.

2.6.2 Improvement Heuristics

We propose two improvement heuristics that enhance an existing solution. These heuristics have the same assumptions as the constructive heuristics of Section 2.6.1. They use an existing solution as their input and create a solution with a lower objective value if an improvement is possible. We propose an inter-route and an intra-route improvement heuristic.

The intra-route improvement heuristic enhances the placement and schedule of each MHPC tour independently. The heuristic traverses each of the existing tours of

MHPCs, starting with the first task the tour visits when the MHPC leaves its initial location, and attempts to reposition that task in the same tour and adjusts the task schedule for a local improvement, i.e., this heuristic does not touch the assignment component of the solution. The heuristic does this by trying all possible re-positioning of the specific pick up or delivery of the task (less than $2n$ possibilities), and matching it with the best possible schedule if scheduler is non-preemptive (less than $n+m$ possibilities) or with the earliest delivery order first schedule if the scheduler is preemptive. If the attempt changes the tour or schedule, the heuristic calculates the objective value again and in case of a lesser value, it changes the specific tour and schedule. It then reconsiders the first task of the new tour after leaving the initial location and repeats the re-positioning attempt. If the attempt does not change the tour or schedule or does not result in an improvement, then the heuristic moves on to the next task in the tour and once it is done with the tour of an MHPC, it moves to the tour of the next MHPC. The heuristic self-terminates if it has examined all tasks and all of the tours. When we use this improvement heuristic, we terminate it after a fixed number of iterations if it has not already self-terminated.

The inter-route improvement heuristic works similarly. It enhances the solution by attempting to change the MHPC assignment of a task. The heuristic traverses each of the existing tours, starting with the first task the tour visits when the MHPC leaves the initial location, and attempts to reposition that task in the tour of another MHPC and adjusts the execution schedule for a local improvement. The heuristic does this by trying all possible placements of the specific pick up or delivery of the task (at most $2n$ possibilities) in the new tour, and matching it with the best possible schedule if scheduler is non-preemptive (less than $n+m$ possibilities) or with the earliest delivery order first schedule if the scheduler is preemptive. It also updates the assignments and the tour of the MHPC to which the task used to be assigned to with its deletion. The heuristic calculates the objective value after this attempt

and in case of a lesser value it changes the assignment and the corresponding tours and schedules. It then moves on to the next task in the tour and once it is done with the tour of an MHPC, it moves to the tour of the next MHPC. The heuristic self-terminates if it has examined all tasks and all of the tours. When we use this improvement heuristic, we terminate it after a fixed number of iterations if it has not already self-terminated.

2.6.3 Applying Heuristics to Solve Variants of MHPC Scheduling Problem

In this section, we discuss how the heuristics of Sections 2.6.1 and 2.6.2,can be applied to the variants of MHPC Scheduling Problem.

To do this, we go over the list of MHPC Scheduling Problem variants again and describe the correct choice of heuristics for each variant:

- **Multiple Processors:** Both heuristics are proposed for a general case of multiple processors. A single processor problem will a special case of these with number of processors equal to 1. We can use the heuristic of Section 2.6.1 and combine it with an improvement step using the heuristic of Section 2.6.2.

- **Multiple Tasks per User Node:** With multiple tasks, all the above heuristics are applicable. It suffices to define a distinct pick up and delivery version for each task of a location in that case. Travel times among different pick up and delivery in the same location will be 0 in this case.

- **Task Availability Times:** In all other variants of MHPC Scheduling Problem, we have considered waiting times only at the delivery locations, i.e. that we only have to wait when the moving vehicle is at a delivery location but the task to be delivered has not yet finished processing. In this variant, waiting may also be required at pick up location, if the task is not yet available. Then the heuristics are applicable with this consideration and will still attempt to minimize both

travel times and these generalized waiting times. Current implementations of all heuristics allow for definition of any availability times for tasks.

- **Scheduling Overhead:** Scheduling overhead refers to the time that is accounted for preempting a task for another or for scheduling a new task on a processor after the previous is finished. Note that in the non-preemptive case, it suffices to add $n\times$ number of the tasks $\times$ new task overhead to the running time to consider the overhead. In the preemptive case, the preemption overheads need to be added with more consideration. Current implementations allow for addition of such overheads as well.

- **Objective Values:** All of the above heuristics build a tour greedily or improve it in the direction of a given objective. We have allowed both TT and ATC to be chosen as the direction of building or improving a tour, and addition of any other objective seems straightforward.

It must be emphasized that both heuristics are sub-optimal because they build a tour greedily and at each step they do not consider the future, or equivalently, it does not consider that changing the past decisions that may result in an improved solution.

2.7 Online MHPC Problem

In this section, we turn into the online version of the MHPC problem. This is a generalization of the problem introduced in Section 2.6.In this case, we assume that knowledge of the tasks arriving is only revealed to the MHPC Controller and hence the corresponding MHPCs only after the User Node has requested service. We are interested in a *complete schedule*, defined as one in which all the tasks have been fully executed and their results delivered.

Before we develop a proper heuristic for this problem, we turn our interest to the

proper choice of an objective value. Since in the online problem tasks continuously arrive over time, RT loses its meaning, as there is no "last task" that the MHPCs have to serve. There are a number other of possible optimization metrics among those introduced in Section 2.2.3 for this problem, though. Among them, we have chosen NAF which is a metric that can be properly applied to the scenario when an unknown number of tasks arrive over time. A closer look into the value of NAF defined in Section 2.2.3 shows that it considers the interest of user nodes in its numerator by trying to deliver the tasks as soon as possible while being fair by applying a normalization in the denominator to not prioritizing a long task over many short tasks[7].

2.7.1 Base Online Algorithm

In this section, we present the high-level view of the MHPC scheduling algorithm. The online instance is described above. To solve such online instance of the MHPC problem, it suffices that we repeatedly solve the offline problem any time a (group of) task(s) arrives. To see this, we introduce Algorithm 2 as follows:

The online algorithm uses the task information that is already revealed to obtain an initial solution. It then repeatedly "cuts" the solution (see Section 2.7.2) when new tasks arrive, recalculates the solution for the portion of the current solution that is not served after this revealing moment, as well as the newly arrived information, and "merges" this new solution with the preserved portion of the solution (see Section 2.7.3). Note that the online algorithm assumes that the time taken to calculate the *New Solution* using Algorithm 2 is negligible compared to the time that it take to process the tasks; otherwise, one can add this computation time as a function of

[7]The algorithms described in this section and Section 2.6 are capable of replacing any objective value fitting other scenarios without affecting the behavior of the algorithm. Some other possible objective values of interest include RT (the time that the latest MHPC finishes the processing of its last task), average completion (the average time between pick up and delivery of all tasks), and average waiting time for all tasks.

Algorithm 2: Online MHPC heuristic pseudo-code.

Input: Groups of tasks: (t_i, G_{t_i}), Number of MHPCs/processors per MHPC: (V, m) , Service period: t_S

Output: *Base Solution* = MHPC Assignment (per instance): $tasks_k \; \forall k \in V$, MHPC Tour (per MHPC): $tour_k = \{\ldots, (\pm i, r_{\pm i}, w_i), \ldots\}; \forall k \in V$, Task Execution Schedule (per MHPC) $sched_k = \{((t_1, t_2), i)\} \; \forall k \in V$

```
Initialization: Use Algorithm 1 to solve for the tasks available at time 0 ;
Store this solution as Base Solution. ;
while time < t_s do
    for Each task group (t_i,G_{t_i}) do
        Use Cutting Algorithm to cut the solution at t_i;
        Store the portion of the base solution before t_i and store it in Base
        Solution. ;
        Merge the portion of the solution after t_i with the tasks in G_{t_i} ;
        Use Algorithm 1 to solve for the merged input ;
        Store this solution as New Solution ;
        Properly Merge New Solution into Base Solution
    end
end
return Base Solution ;
```

the size of the problem and slightly modify the behavior of the MHPC Controller to address this. Our implementation allows for batching the groups of tasks in these cases. This batching will require the MHPCs to recalculate the solution only after either a specific amount of time has passed since the arrival of the first group of newly arriving tasks or a specific number of tasks have arrived. Batching is usually only needed to avoid frequent re-calculations of the solution. In our implementation batching is given and implied, i.e., we do not focus on the best practices to perform the batching and rather assume that revealing of tasks in groups rather than individual tasks is due to some batching that is underway.

2.7.2 Cutting Algorithm

As described in Algorithm 2, we need to cut the solution at the moment that a new group of tasks arrive. This action involves keeping the parts of the solution on which the MHPC has already done some work and re-planning for tasks that are not yet

served or are partially served along with the new group of tasks. A task, in this framework, goes over the following phases that are depicted in Figure 6 as well:

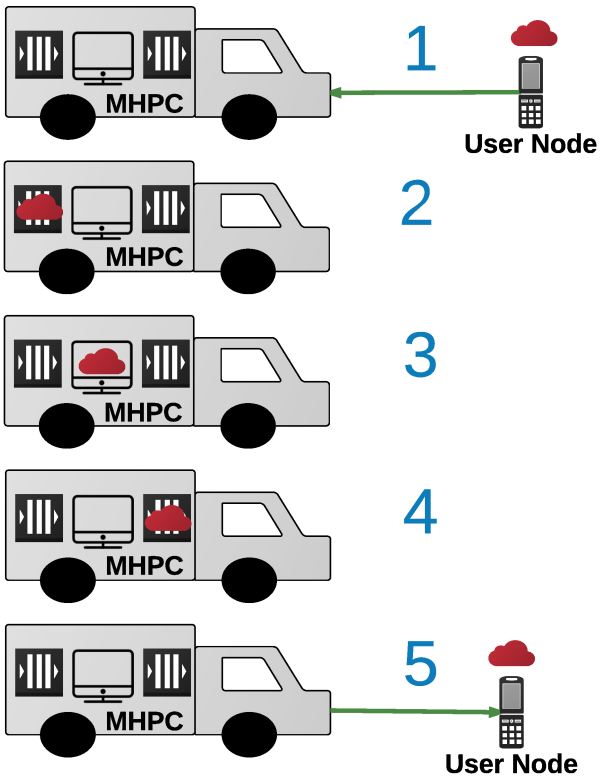


Figure 6: Life cycle of a task in the MHPC framework: (1) The task initially resides on user's hand-held device, (2) It is picked up by the MHPC and awaits processing, (3) It receives processing at the MHPC, (4) It awaits delivery at the MHPC after completion of processing, (5) The result is delivered back to the user node.

1. It may not even be picked up by the time that the new information arrives. In this case, the task is treated as another piece of new information that has arrived and must receive full pickup, processing and delivery service. This is phase 1 in Figure 6.

2. It may be picked up, but its processing may not have yet started by the time that the new information arrives. In this case, the task needs processing and delivery services. This is phase 2 in Figure 6.

3. It may be picked up and partially processed by the time that the new information arrives. In this case, the task needs to be processed for the remaining

portion of it and then needs to be delivered. Note that the remaining processing is re-planned by the controller and does not necessarily occur immediately after the arrival of the new information as a continuum of the previous processing. This is phase 3 in Figure 6.

4. It may be picked up and completely processed. In this case, the results need to be delivered properly to complete the service on the task. This is phase 4 in Figure 6.

5. It may be completely processed by the time that the new information arrives. In this case, the task does not need any further service and "cutting" has no effect on it. This is phase 5 in Figure 6.

To explain the operations involved in cutting the solution at the moment t_i when a new group of tasks arrives, assume, for the tour of k^{th} MHPC that that t_k is the moment that the MHPC finishes serving (i.e., delivering all the result) the last task among those for which it has been planning since the last period.

- If $t_i \geq t_k$, we assume that each MHPC has stopped at the location of delivery of its last task. When planning for a new group of tasks, we assume that these are the new initial locations and use Algorithm 2. All of the solution in the previous step is marked as *served* and Algorithm 1 solves the problem only for the newly arrived groups of tasks to deliver the *New Solution*.

- If $t_i < t_k$, we need to re-plan for any pick up, processing and delivery that has occurred after t_i. To do this, we need to find the location that each MHPC will be at time t_k. At this moment, the MHPC will be either traveling between two user nodes and its location can be found using interpolation or it will be finished with the last task and its location will be the location of that task. To cut the solution at t_k, we shall keep tasks of the first four categories described above

and treat them accordingly for the completion of service on them, whether it is pick up, processing, delivery or a combination of those. We then add the new group of tasks to this list of *non-served* tasks and use Algorithm 1 to deliver the *New Solution*. Note that any of the tasks that remain after cutting the solution will be marked as *served*.

2.7.3 Merging Solutions

This operation merges the *New Solution* that results after applying Algorithm 1 with the *Base Solution*. This merging shall account for the offset the time that the new information is revealed, t_i imposes. The merged solution will have the same format as described for any solution in Section 2.2.2 and will be used as the Base Solution in the next iteration of the for loop in Algorithm 2. This will provide a complete solution to the problem.

2.8 Evaluation

In this section, we perform various experiments to understand the performance of the algorithms presented in Sections 2.6 and 2.7. We start by reviewing examples of simple instances of the offline problem to understand the structure of the solutions. Then we move forward to understanding the offline algorithm and how its performance compares to the formulation of Section 2.4. Finally, we present various experiments that study the online heuristics that solve a more realistic instance of the problem.

2.8.1 Examples of the Offline MHPC Problem

In this section, we study some examples of performance of Algorithm 1 for both preemptive and non-preemptive scheduling. While this gives us a preview of components of the solution, discussed in Section 2.2.2, it also helps understand the difference in behavior of the algorithm in case of preemption and how it may benefit the solution.

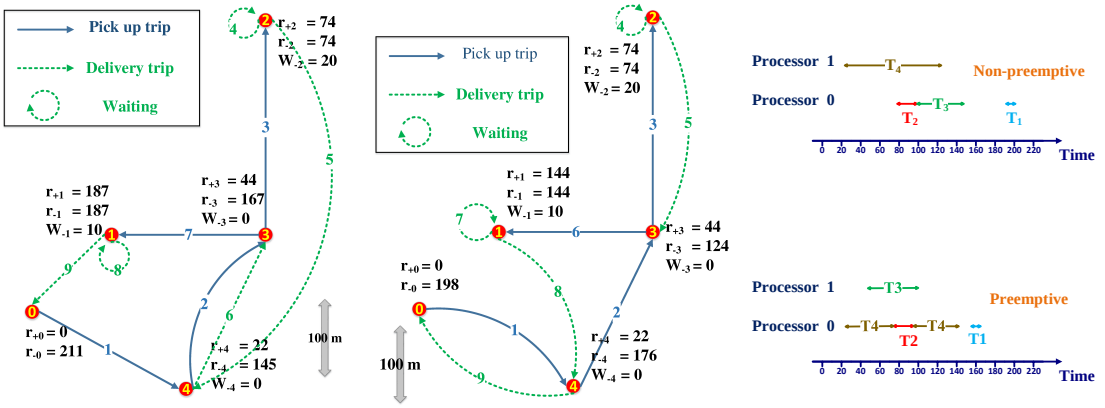
Our experimental setting for this section is depicted in Figure 7a. Assume that

a two processor MHPC is initially at location 0, and there are tasks of duration $10, 20, 50, 100$ at user nodes $1, 2, 3$, and 4 respectively all known to the MHPC at the beginning of time.

Since there is a single MHPC in question, we do not have any MHPC assignment in the provided solutions. Figures 7a and 7b show the MHPC Tour and Task Execution Schedule for the instance described above using a non-preemptive and a preemptive scheduler respectively in a pictorial manner. Note that the tour specifies visit times, r_{+i} for pick up and r_{-i} for delivery, and visit durations, in the form of waiting time before delivery, W_{+i}. It is evident from the figure that even though for the shorter tasks of T_1 and T_2 a better strategy is to pick up, process and deliver results back without moving, for longer tasks the power of computation while moving leads to saving some time. Also, as expected, the non-preemptive solution takes a total of 211 while the preemptive takes a total of 198, confirming the assumption that preemption can make the operations more efficient. To complete this example, we have shown the Task Execution Schedule component of the solutions in Figure 7c. Note that as expected for the non-preemptive case, each task is scheduled only once, while for the preemptive, tasks may be sub-divided and scheduled over different intervals and/or on different processors.

2.8.2 Validation of the Offline MHPC Heuristic

In this section, we run a simple example to compare the offline heuristic of Section 2.6 with the solution obtained from solving the formulation of Section 2.4 using CPLEX [2]. With a similar experiment setting to the one described in SEction 2.8.1, where we use a non-preemptive scheduler and single processor MHPC, and we assume all tasks are available from the start. We keep all four tasks equally long and change their time duration from $1s$ to $100s$. As shown in Figure 8, the objective value, which is RT in this case for both heuristic and the exact solution increases with larger tasks,



(a) MHPC Tour / non-preemptive Scheduler. **(b)** MHPC Tour / preemptive Scheduler. **(c)** Task Execution Schedule.

Figure 7: MHPC Tour and Task Execution Schedule of a sample solution of an MHPC problem with a two processor MHPC serving 4 user nodes. Tour starts from location 0 and can be followed using the numbered arrows. r_{+i}, r_{-i}, W_i indicate the pick up, delivery, and waiting times of the single task at location i. Each schedule shows intervals that either of the two processors of the MHPC are busy processing a task.

with the exact solution outperforming the heuristic modestly at some of the points. Once the tasks get longer than $53s$, both the heuristic and the exact solution have objective values increasing linearly. This occurs due to the fact that for long tasks, the lower bound becomes dominant in the objective value. In fact, it can be shown that for the single processor non-preemtpive example after this point, the time to return to depot exactly equals to $\min_i t_{0i} + \min_i t_{0i} + \sum_{i=1}^{n-1} T_i$, where min_1 and min_2 are the first and the second minimum of the given vector. We have repeated this experiment for various smaller examples and for multiple processors, and the results have been similar in all cases. We expect the gap between exact and heuristic solutions to be larger for bigger examples due to the fact that the heuristic is built greedily and an early deviation from the optimal solution propagates into the further steps of the heuristic.

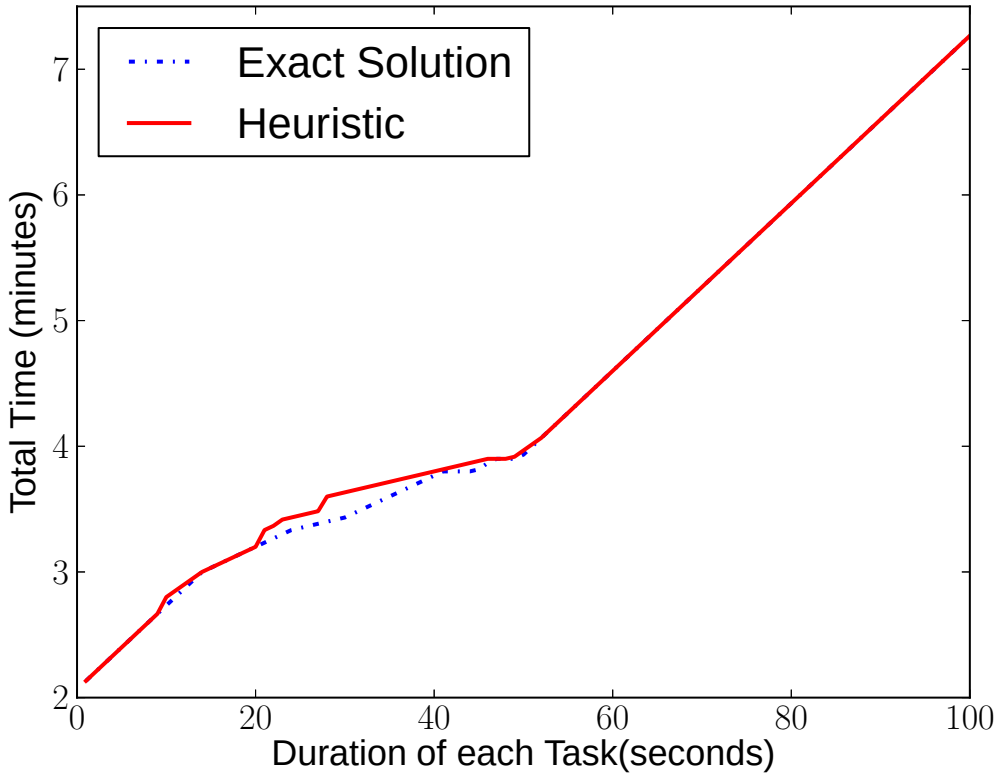


Figure 8: Comparison of the exact solution and the heuristic for a non-preemptive single processor MHPC serving 4 tasks.

2.8.3 Performance of the Offline MHPC Heuristic versus Number of Processors for Non-preemptive and Preemptive Schedulers

In this section, we explore the effect of number of processors on RT as the objective value of examined instances to understand the advantages of preemption in multiple processor cases by comparing the preemptive and non-preemptive cases. In this experiment, we use 25 task locations, including a depot location representing the initial location of our single MHPC and deploy them in a $10km \times 10km$ field. To do so, we use a Clustered Gaussian Model(CGM), in which we deploy, almost equal number of users around some hotspots according to a Gaussian distribution. We manually place four hotspots, reasonably far from each other. To keep the settings reasonable, we do not allow the creation of a node in a $100m$ radius of each created node. Note that if two nodes are too near to each other, namely within the wireless transmission

range, then having a trip from the location of one to the other might not be the wisest strategy for task processing and instead the vehicle can park in an area that is within the range of both user nodes to pick up or deliver tasks to/from them. This case will effectively be equivalent to having a single user node with two tasks. We also derive the duration of the tasks from a similar one dimensional CGM, with means around the values of $10s$, $25s$, $50s$, $100s$, $200s$, $600s$, $800s$, $1200s$ and $2000s$. This guarantees that all sorts of short and long tasks are present in the experiment. Finally, we assign an availability value to each task durations from a uniform distribution between 0 and 300 seconds (5 minutes). These availability values make the scenario similar to the online case that tasks arrive over time, except that to keep the offline nature of instances in these experiments, we assume all these availabilities are known at time 0. In both experiments we run the construction heuristic of Section 2.6.1 followed by an improvement step as suggested by Section 2.6.2.

We change the number of processors from 1 to 10 and observe the total completion time for the improved version of non-preemptive and preemptive heuristics. Figure 9 shows this effect for the both experiments. We observe that as we increase the number of processors the completion time almost flattens for both types of schedulers when we have 8 processors. This is due to the fact that with the given tasks, extra processors will not be needed. It is also notable that preemptive scheduling results in lower objective value, translated to better performance for the preemptive case. This is also intuitive as in general, non-preemption can be viewed as a special case of preemption that the algorithm can choose to adopt if it results in a better objective value. It is notable that the advantages of preemption are generally more evident with lower number of processors.

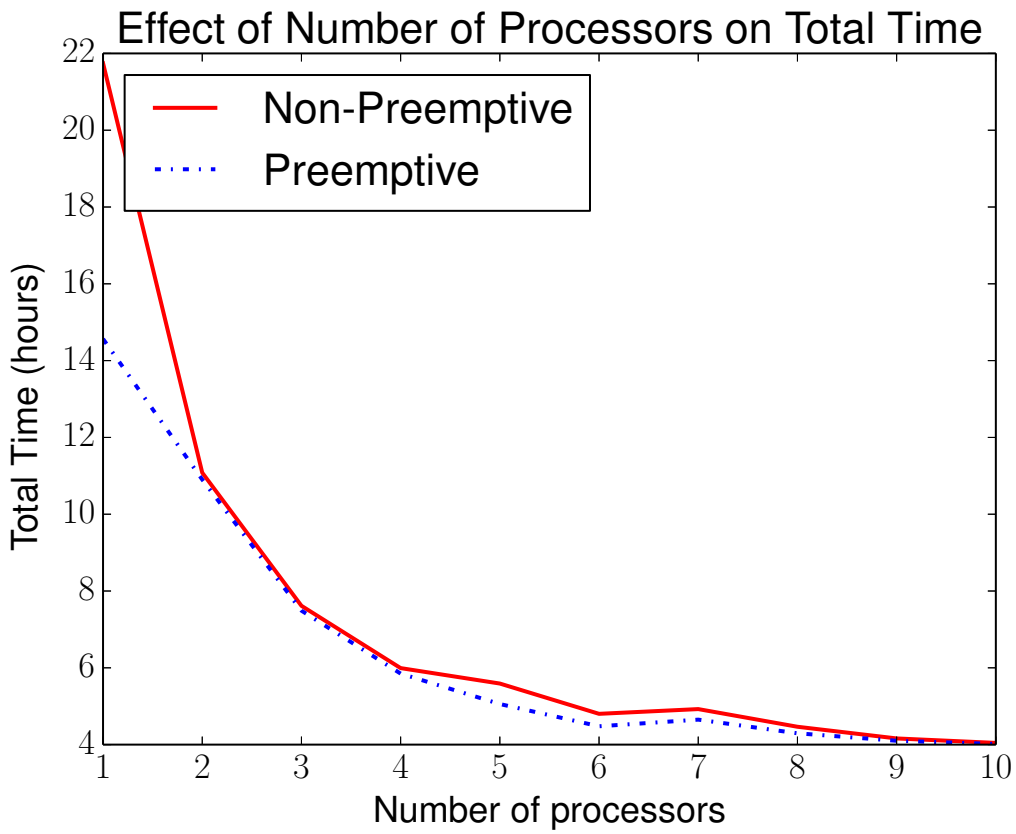


Figure 9: Comparison of the preemptive and non-preemptive scheduling for an MHPC problem. The x-axis shows behavior of each scheduler as the number of processors grow. The y-axis shows the objective value RT in hours.

2.8.4 Performance of the Offline MHPC Heuristic versus Preemption Overhead

In this section, we explore the effects of preemption overhead. In a realistic scenario, on a preemptive scheduler, once a task is decided to be preempted, the context switch will incur an overhead. This overhead can be more than the overhead incurred by replacing a new task for a finished task. We use the same settings as Section 2.8.3 with 2 processors. We assume a new task overhead of 0, as it affects both preemptive and non-preemptive variants the same way, while we vary the preemption overhead. As shown in Figure 10, the preemptive scheduler does better than the non-preemptive the overhead becomes nearly 140 seconds. Notice that in the figure and after this value we have many points that the preemptive scheduler's completion time crosses

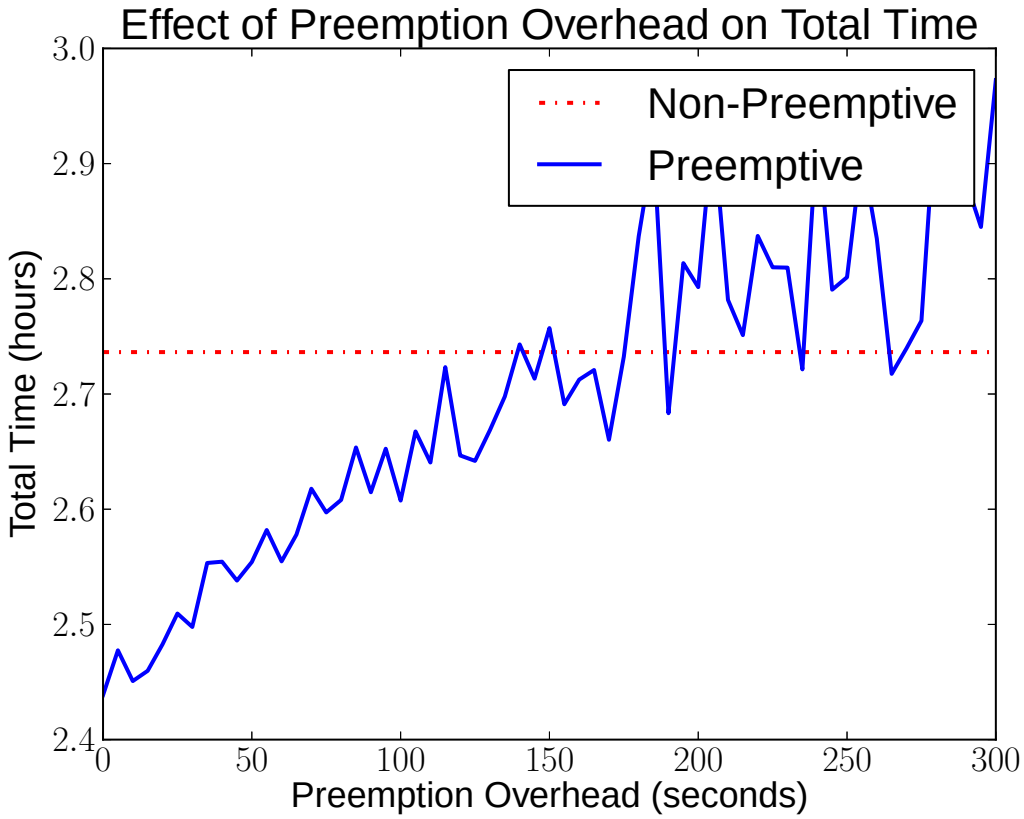


Figure 10: Effect of preemption overhead on a preemptive MHPC. The dashed line shows the non-preemptive problem as baseline and the solid line shows the behavior of the MHPC problem asthe preemption overhead grows on X-axis. The objective value, RT, is used for comparison along Y-axis.

the non-preemptive. This is due to the fact that with large overhead values few more preemptions affect the running time significantly. In an optimal solution one would expect that with a larger preemption overhead there is a low cost solution, the same can be used with a slightly smaller preemption overhead, but since the heuristic constructs the solution greedily such assumption does not hold and the behavior is not surprising. In the rest of this thesis, we ignore the preemption overhead for simplicity and to focus on studying the effects of other parameters.

2.8.5 Performance of the Offline MHPC Heuristic versus Task Availability

In our last experiment with the offline version of the MHPC heuristic, we explore the effect of having tasks available at a later time on the completion time. This is interesting to later contrast with the online case. Having task availabilities makes a scenario very similar to the online case, except that the MHPC Controller has advance knowledge of when new tasks will be revealed to it and what the value and location of those tasks will be. We assume the same settings as Section 2.8.3 except that we set the means for the derivation of the task durations to slightly smaller values of $5s$, $10s$, $25s$, $50s$, $100s$, $200s$, $500s$ and $600s$ and an MHPC serving 7 user nodes each having 3 tasks as an example of the MHPC scheduling problem with multiple tasks per node.

To study the effect of task availability times, we derive the inter-arrival times between the availability of each of the 21 tasks from an exponential distribution. We change the scale parameter (inverse of rate) from 1 to 20. Each time we change the distribution to derive the task availability values from, we average over 10 runs to alleviate the effect of randomness of the distribution.

As shown in Figure 11, making tasks available at a later time makes the total time slightly higher, but the heuristic is capable of accommodating the availability to suggest a tour that serves the earlier tasks first. This is also due to advance knowledge of these information to the MHPC Controller. We do not expect such behavior in the real online case where arrival and metadata of these arriving task are not know apriori.

2.8.6 Performance of the Online MHPC Heuristic versus Task Arrival Frequency

In this section, we study the effect of the frequency of task arrival on the system. We turn into a more diverse model for instance generation in this case. To disperse

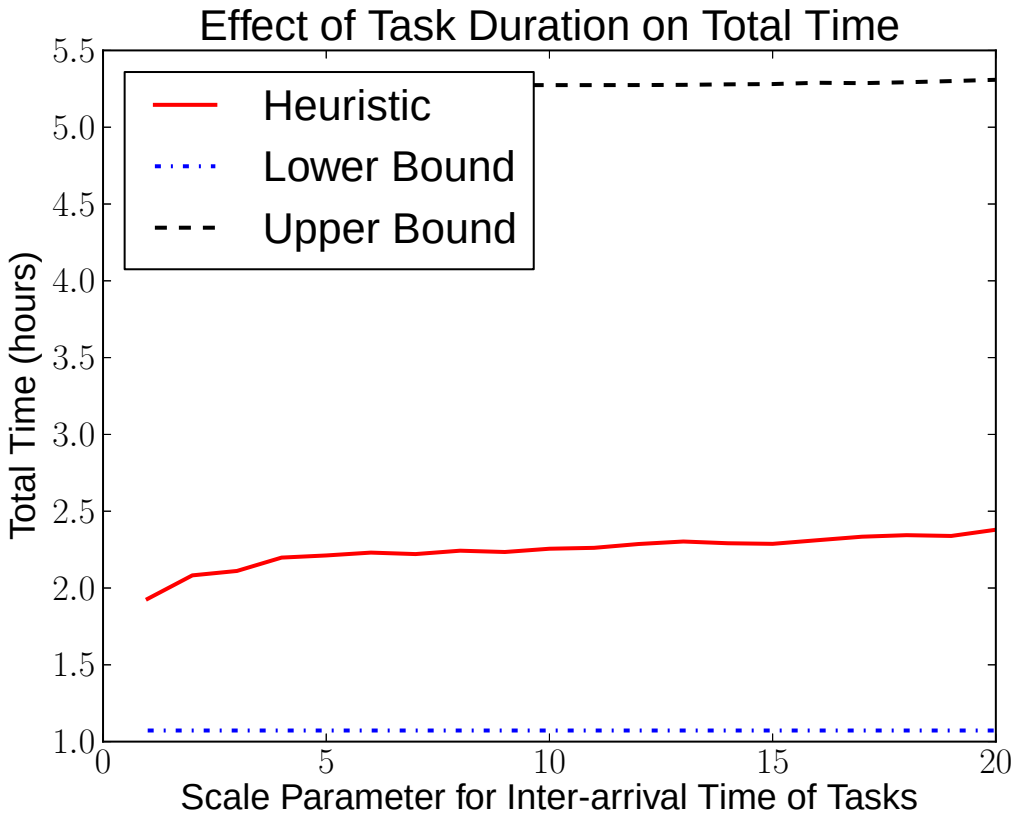


Figure 11: Effect task availability on the performance of a quad-processor MHPC system. An upper bound and lower bound for the solution is shown according to the descriptions of Section 2.5.1. This example concerns averaging performance of offline instances of the problem as tasks become available more sparsely into near future. X-axis shows the scale parameter for the exponential process governing inter-arrival time of task availabilities. Y-axis shows the objective value, RT, in hours.

the task locations, we divide the area into zones, as seen in Figure 15, some of these zones correspond to the areas in the field that users cannot access, e.g. unreachable mountains, areas with severe weather, contaminated areas, etc. To account for this, with some probability (set to 80% in our experiments), we mark each zone as usable. For each usable zone, we disperse a number of locations, drawn from a uniform distribution with a mean of $n_z \simeq \frac{n}{|Z_u|}$, where n is the total number of task locations and $|Z_u|$ is equal to the number of usable zones. In all the evaluations that will follow, we assume that there are 10 such task locations.

We run the system with 3 MHPCs serving 600 tasks arriving over time. Each

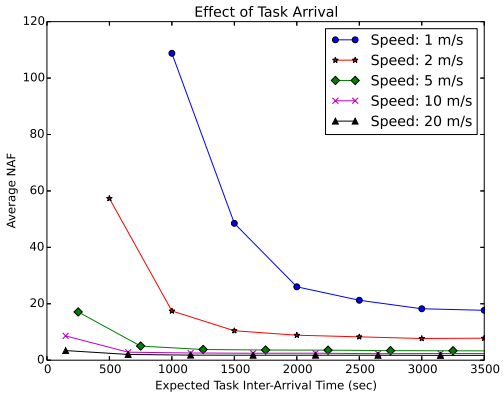
task has a duration around one of the cluster centers of $30, 60, 120, 240, 360, 600, 900$, and 1200 second plus a small Gaussian error. This represents various job types that MHPCs have to handle. Tasks are available for pick up after notification at a time exponentially distributed with a mean of 60 seconds and they do not have deadlines. We run the test for MHPCs moving with speeds of $1, 2, 5, 10$, and $20m/s$, 20 times for each speed and report averages. We change the expected inter-arrival time of tasks from a minimum value respecting the capacity of the system to 3500 seconds in 500 second steps[8].

Figure 12a shows the average NAF (see Section 2.2.2 for definition) value for various speeds. It shows that both increasing the speed and increasing the inter-arrival time of tasks decreases the cost. However, in the latter once the MHPCs have enough time to serve all the tasks that arrive before new tasks arrive, their behavior and thus the cost remains constant. Figure 12b shows the utilization of all MHPCs, defined as average time that one MHPC is using its processors. The value of the utilization is the same for all speeds, since task durations are about the same and the only difference in behavior of MHPCs under various speeds is that at higher speeds there is more idle time and less travel time and vice versa for lower speeds. The utilization, however, decreases as tasks arrive less frequently in all cases due to the fact that there is more idle time for the MHPCs. These figures collectively show that while having less frequent tasks allows the MHPCs to serve them better, this will result in more resources being wasted.

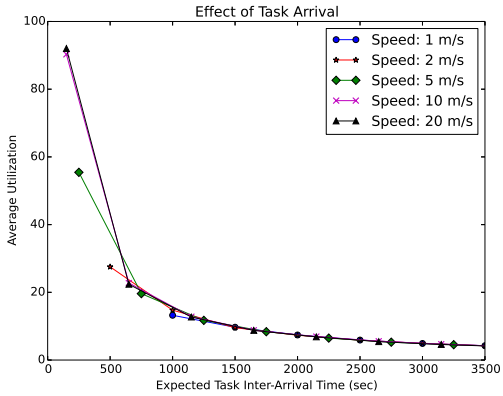
2.8.7 Performance of the Online MHPC Heuristic versus Size of Task Groups

In this section, we study the effect of the size of groups of tasks as they arrive. This can represent the effect when more tasks are batched together. Such a case might

[8]The minimum arrival rate used for the speeds are one task every $1000, 500, 250, 150$, and 150 seconds respectively for lower to higher speeds



(a) The Objective Value, average NAF.



(b) Average Utilization of the MHPC processors.

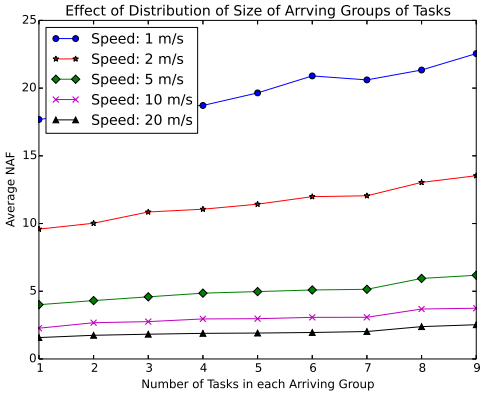
Figure 12: Effect of arrival frequency of tasks on the performance of online MHPC. X-axis shows the mean value for exponential process governing inter-arrival time of tasks. Tasks arrive more sparsely along this axis. The metric measured on Y-axis is noted o the corresponding captions.

happen if, for example, the FOBs perform some internal batching to inform the MHPC only after a certain amount of time or the collection of a certain number of tasks. To respect the system capacity, arrival rates are chosen to be $10000, 5000, 3000, 2000$, and 1500 for speeds of $1, 2, 5, 10$, and 20 m/s respectively. All other settings are similar to Section 2.8.6. We change the size of the task groups from 1 to 9 tasks.

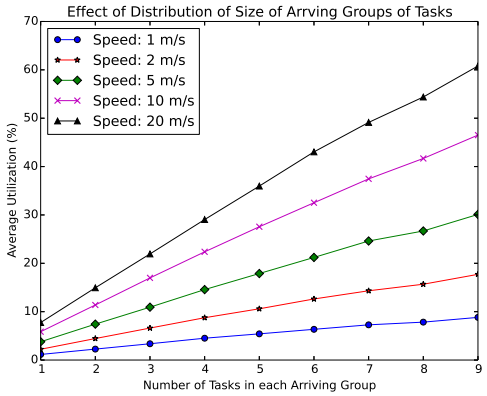
As Figure 13 shows, it is observed that with larger groups, there is more load on the MHPC, making it less effectively serve the tasks while utilizing more of its processing power.

2.8.8 Performance of the Online MHPC Heuristic versus Task Deadlines

In this section, we study the effect of deadlines. A standard test will be to utilize a scenario similar to Section 2.8.6 and vary the *deadline margin* to see how it affects task rejection, as well as cost. To be fair among different task durations, we define deadlines for each task as "task revealing time + task duration + deadline margin". Here, task revealing time is the moment that MHPC Controller is notified of the arrival of the new task. We change this margin in 500 increments until it is 7500



(a) The Objective Value, average NAF.



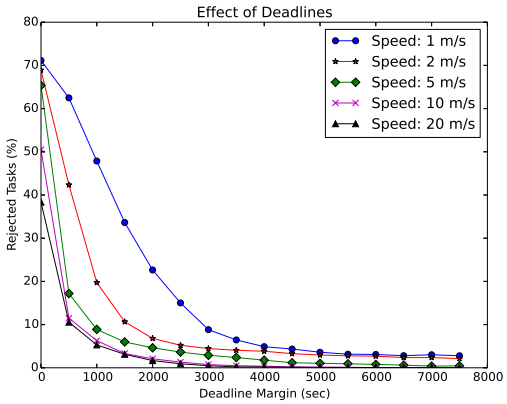
(b) Average Utilization of the MHPC processors.

Figure 13: Effect of number of tasks arriving in each batch on the performance of online MHPC. X-axis shows the mean number of tasks arriving in each group. The metric measured on Y-axis is noted o the corresponding captions.

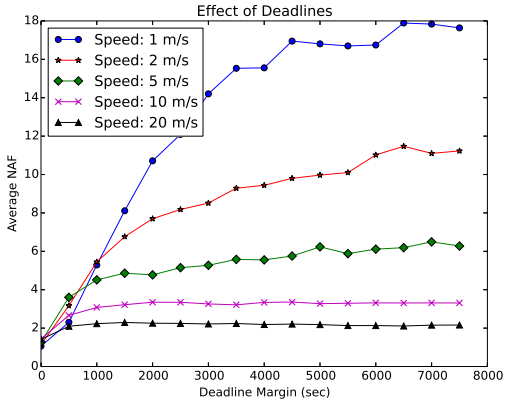
seconds. Figure 14a shows that in such a test, with higher deadline margin, there are fewer rejected tasks since the MHPCs will have more time to deliver results by the given deadline. Figure 14b shows that while the cost initially increases, when there are fewer task rejections, the cost remains steady, as the MHPC will deliver the same service and the extra deadline margin will not be helpful. Note that the reason that in the region of very small margins the cost for speeds of 1 and 2 are better than speed of 5 is due to the fact that in these small margins the high percentage of rejected tasks by the slower MHPCs results in a better service to the few remaining tasks.This, however, does not mean that the system is giving better service with respect to all tasks though. These figures show that while some deadline margin will help the MHPCs to serve the tasks instead of rejecting them, there is no necessity for very large margins.

2.8.9 Examination of Effects of Travel Distances on the Online MHPC Heuristic

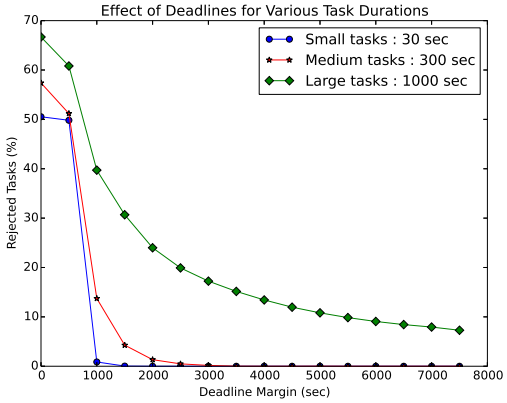
In the last three experiments that we perform on the MHPC scheduling algorithm, we turn our attention into simple experiments that can reveal more understanding



(a) Average percentage of rejected tasks.



(b) The Objective Value, average NAF.



(c) Effect of deadlines on task rejections with task duration.

Figure 14: Effect of task deadlines on the performance of online MHPC. X-axis shows deadline margin of the tasks defined in Section 2.8.8. Deadlines become looser along this axis. The metric measured on Y-axis is noted o the corresponding captions.

about the behavior of the system and aid in design choices.

We perform these evaluations with some simple tests designed to understand the

behavior of the algorithm. The baseline for these tests are as follows: task locations are chosen to be at the center of one of the 8 numbered zones in the 5km×5km field shown in Figure 15. Pick up and delivery locations of each task are the same, aka task locations. MHPCs move with an average speed of 10 m/s. All task durations are the same and they are immediately available after the MHPC is notified and do not have a deadline. Tasks arrive in groups of 1 generated by a Poisson process. For the experiments that follow we generate 600 tasks with a 1000 second mean inter-arrival time. Note that the inter-arrival time between these notifications shall not exceed the system capacity, i.e. tasks shall arrive in a way that the MHPCs can eventually serve them; otherwise delivery time of results will keep increasing and the system cannot serve the tasks. A very rough experimental estimate for a proper arrival rate is $\frac{1}{(n_i/V)T_i+d/\nu}$ where n_i is the average number of tasks in each group that arrives, V is the number of MHPCs, T_i is the average duration of the tasks, d is the dimension of the larger side of the rectangular field, and ν is the average speed of the MHPCs. Arrival rates lower than this number will result in MHPCs serving the tasks with less delay.

The first experiment investigates the effect of the distance between task locations on the performance of the algorithm. We assume a single vehicle serving the baseline system, initially placed at the center of zone 1. Tasks are generated at two locations. One of the locations is always at the center of zone 1, the other location is located at center of zones $8, 5, 4$, and 1 respectively[9]. We repeat the tests 10 times and for three task durations: small tasks of 30 seconds, medium tasks of 300 seconds, and large tasks of 1000 seconds duration.

As Figure 16 suggests, making the locations of task generation farther apart always results in worse performance as the MHPC has to travel longer distances. This

[9]In the case that both task locations are at the center of zone 1, we move each about 150 meters from the center diagonally to keep two distinct locations.

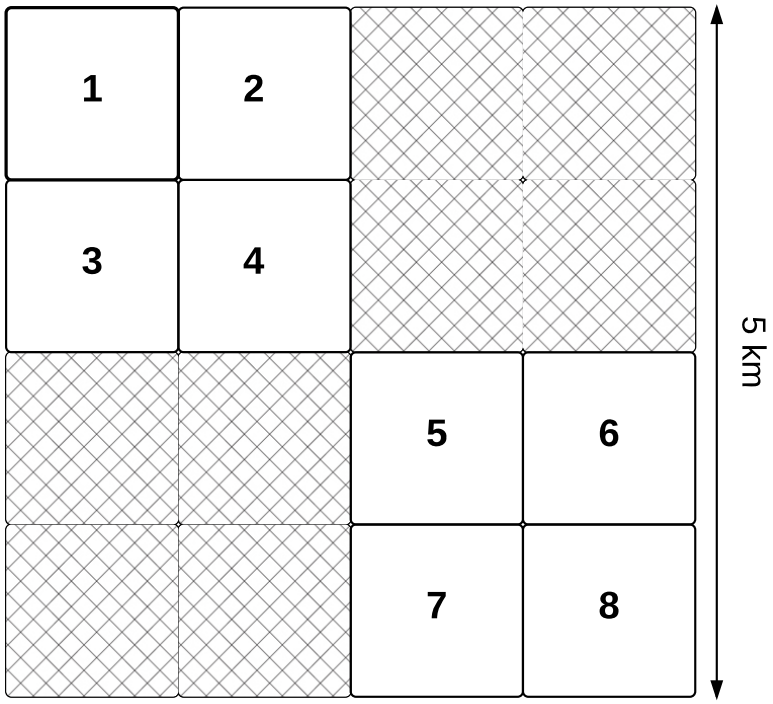


Figure 15: Framework for testing various effect on the performance of the online MHPC heuristics in Sections 2.8.9, 2.8.10, and 2.8.10. Tasks can be generated at each of the numbered locations as described in each experiment.

difference is more evident in the case of smaller tasks, as the MHPC will process the tasks right after pick up and going back and forth between two far locations degrades its performance. It is also notable to see that the value of NAF is smallest for medium tasks. To understand why, one shall notice that the cost for small tasks are higher, because of the normalization by the task duration which is essential for fairness when considering mixture of tasks. For larger tasks the time difference from availability to delivery is much larger compared to task durations and this in turn degrades the performance. This also shows that the MHPCs best serve medium sized tasks, with durations around half the time to travel the diameter.

2.8.10 Examination of Effects of Mobility Pattern on the Online MHPC Heuristic

In this section, we perform another simple experiment to understand how the MHPCs move. To do this, consider the same setting of Section 2.8.9 while the second location

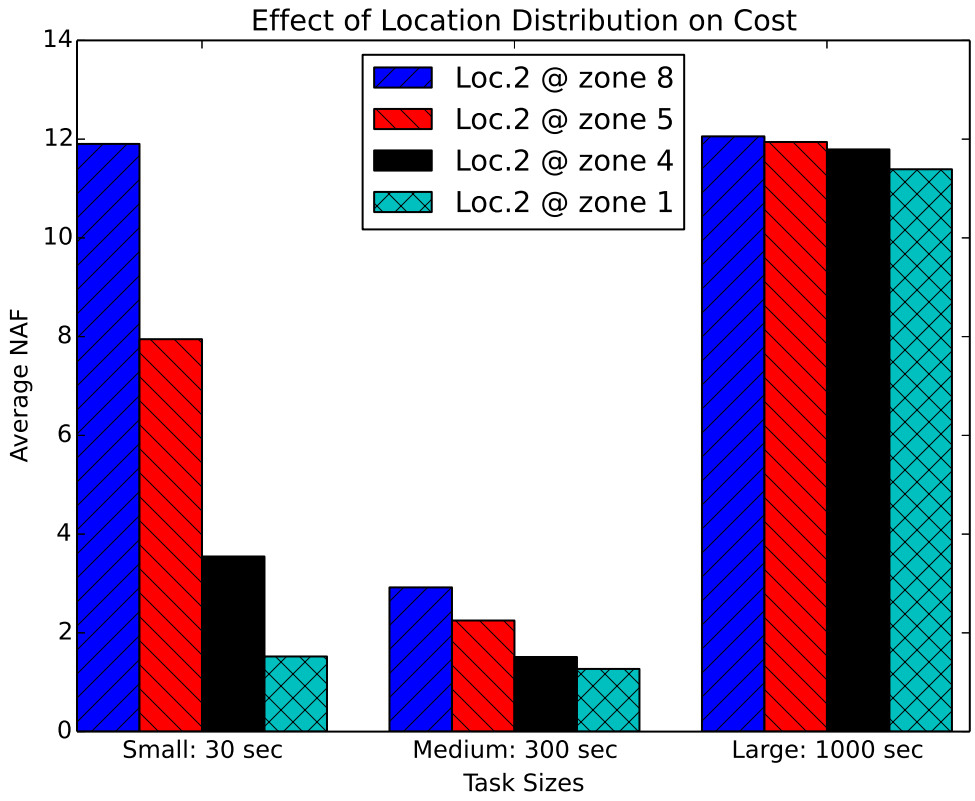


Figure 16: Effect of distance among task locations on the performance of the online MHPC heuristic. Groups of bars on the X-axis shows the experiment for small, medium, and large tasks, respectively. One task is always placed at location 1 in Figure 14. Each group repeats the second task placed at 8, 5, 4 and 1 of Figure 14 respectively. Y-axis measures the objective value, average NAF.

is located at zones $8, 5$, and 4 only. We call the zone where the initial location of each MHPC its "home". We then track the percentage of pick ups done by each MHPC at its home zone. Figure 17 shows the percentage of pick ups done by the second MHPC that are in its home zone. This behavior is symmetric for the other MHPC. It shows that for short tasks, the MHPCs always remain in the zone where they are initially placed. As the task size grows though, each MHPC may move to the other zone as well. This is, again, due to the fact that when the MHPCs are busy processing a task, they will use the power of computation while moving to pick up tasks even in the other zone. While for short tasks, deliveries are mostly immediately followed by pick ups.

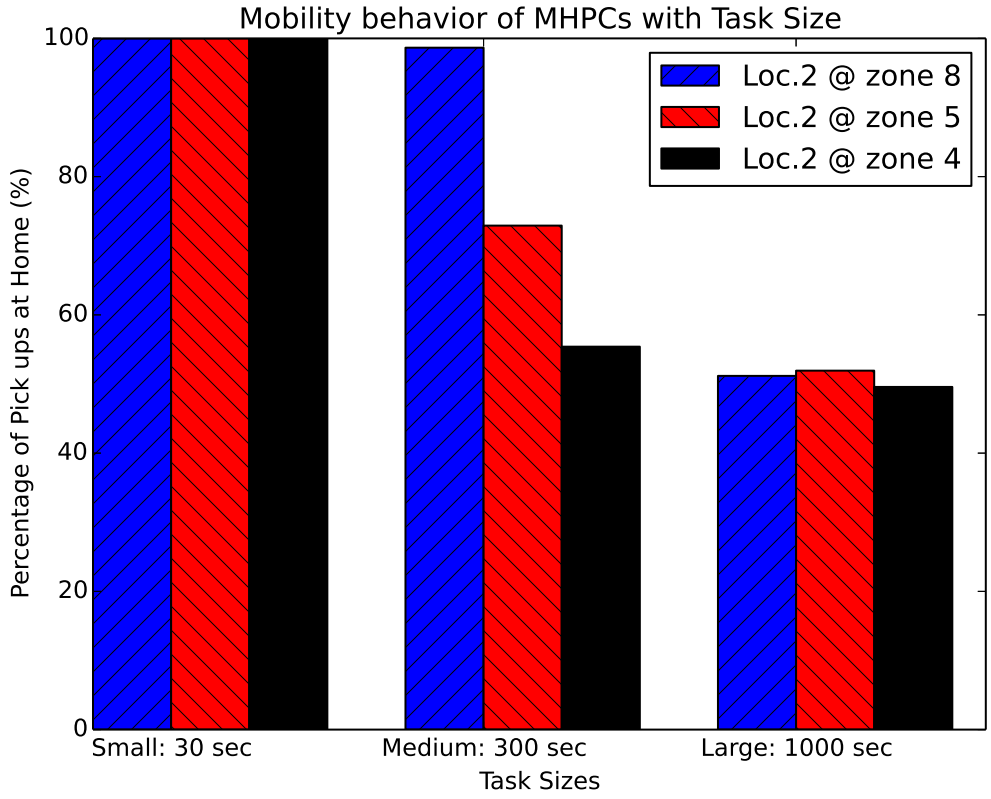


Figure 17: Effects of the mobility behavior of MHPCs on the performance of the online MHPC heuristic. Groups of bars on the X-axis shows the experiment for small, medium, and large tasks, respectively. One task is always placed at location 1 in Figure 14. Each group repeats the second task placed at 8, 5, and 4 of Figure 14 respectively. The framework consists of two MHPCs placed near one of the task locations initially. Y-axis measures the percentage of the time that each MHPC spends serving the nearby task location.

2.8.11 Examination of Benefits of Increasing Vehicles versus Processors for the Online MHPC Heuristic

In this section, we investigate the effect of number of the MHPCs and processors. We started the experiments in this section with settings similar to Section 2.8.9, changing the number of MHPCs and in another experiment the number of processors of each MHPC from 1 to 10. They indicate that increasing the number of MHPCs or processors will decrease the cost as well as the utilization. It was noted that after a point, while there is not much decrease in the cost, utilization continues to decrease. This indicates that while increasing the number of MHPCs and processors

are generally helpful, increase after a point will only waste resources and not provide better service.

Instead of this obvious experiment, we decided to answer a more interesting question: "how does an increase in the number of vehicles compare to an increase in the number of processors?". An answer to this question can be insightful in terms of designing the system, when one can choose between mounting a second or better HPC on the same vehicle or obtaining a new vehicle with a similar HPC.

To demonstrate this, we placed 8 task locations in the 8 shown zones of Figure 15 while the rest of the settings are similar to those of Section 2.8.9. For each of the task sizes, we compare three scenarios: 1 MHPC with 4 processors, 2 MHPCs with 2 processors per each, a single MHPC with 4 processors. While the compute power of these three cases are equal, their mobility is different.

Figure 18 shows that going from 1 MHPC to 2 MHPCs gives considerable improvements in cost, especially when tasks are small or medium size. Dedicating 4 single processor MHPCs does not give comparable improvements compared to 2 dual processor MHPCs. The reason is that in this experiment the task locations $1, 2, 3, 4$ can be considered one cluster and the locations $5, 6, 7, 8$ can be considered another. Having one MHPC serving each cluster can improve the performance but two MHPCs serving the same cluster will not bring significant improvements. It is also noted that these improvements are more evident for small and medium task sizes due to the fact that with larger tasks most of the traveling happens while the MHPC is processing a task and this dominates the time taken to travel for pick up and deliveries. Because of this, we suggest that designers of MHPC-based systems consider employing a strategy of initially positioning MHPCs to serve geographic clusters of mobile users.

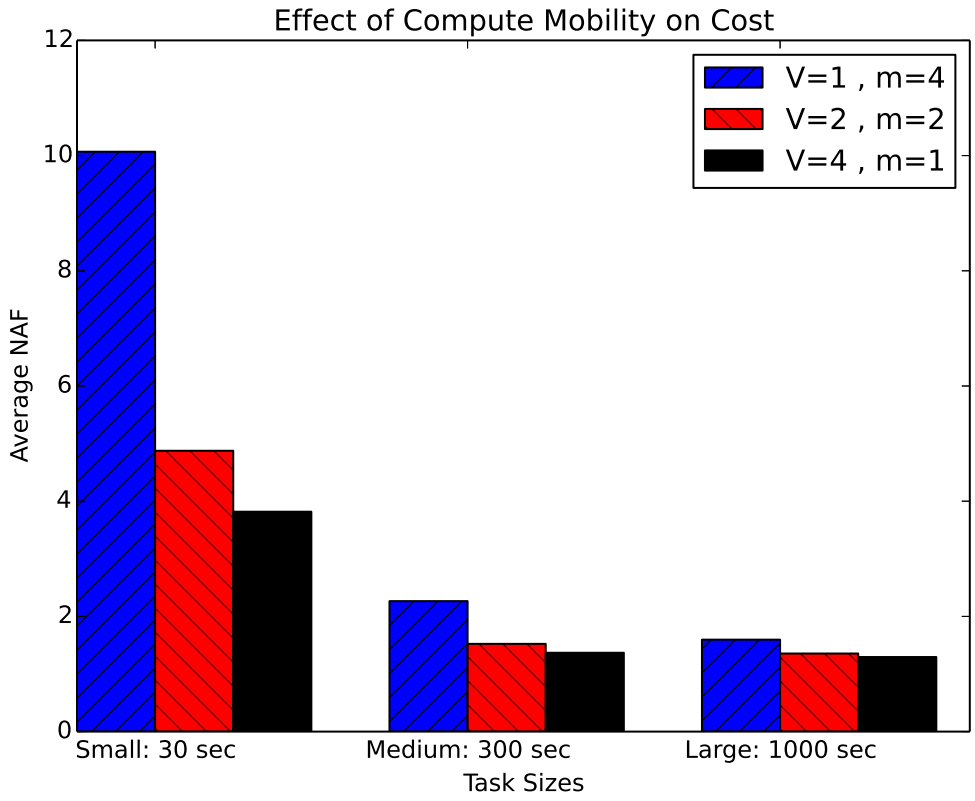


Figure 18: Effect number of MHPCs (V) vs number of processors per MHPC (m) on the performance of the online MHPC heuristic. Groups of bars on the X-axis shows the experiment for small, medium, and large tasks, respectively. Each group compares the three cases of one MHPC with 4 processors, two MHPCs with two processors, and 4 MHPCs with single processors respectively. Y-axis measures the objective value, average NAF.

2.9 Conclusions and Future Work

Motivated by computational offloading requirements in mobile cloud computing, in this chapter, we explored the architecture of a computational ferrying scheme where mobile high performance computers (MHPCs) provide compute resources to wireless user nodes. We studied the problem of scheduling the MHPCs' movement and processing. We started by modeling and formulating the design of such schedules as an optimization problem. We have discussed the challenges of this type of system and have developed algorithms that can be used to produce schedules for larger scale problems. We performed an extensive set of experiments to demonstrate applicability of the algorithms and show the effect of various system parameters on the performance

of such a computational ferrying scheme.

This work represents a first step towards the design and evaluation of the operation of an MHPC-based mobile cloud computing system. There are a number of interesting future directions which include:

- Understanding the robustness of an MHPC system to malfunctioning of its MHPCs and how to design failure recovery mechanisms.

- Enhancements to the model to include variations such as processor-sharing scheduling on the MHPC and the possibility of picking multiple pieces of a computation from different locations or delivery to multiple user nodes.

- Integrating more complicated communication models in the framework, including modeling of wireless channels for both short-range and long-range radio with loss. This might require optimizing the heuristics of Section 2.6 and 2.7 so that the MHPCs start exchanging information with the user nodes once they are at an optimal distance. There is also a requirement to deal with unsuccessful exchange of information due to channel imperfections.

- The consideration of communication among MHPCs either using user nodes as relays or direct exchange of information motivated by the work on multiple message ferry scheduling (e.g., [58]).

CHAPTER III

MESSAGE FERRYING WITH A PURPOSE: SCHEDULING FERRIES TO PROVIDE SERVICE ON A TACTICAL HIGH PERFORMANCE COMPUTER

3.1 Introduction

In this chapter, we introduce the second of the problems proposed in Section 1.1. In this problem, we consider the case where computationally intensive computations are offloaded to a more powerful computing resource located in a stationary High Performance Computer (HPC) that cannot be reached directly by a reasonable communication channel. To communicate tasks to this computation resource, we utilize nodes called Message Ferries (MF) to carry tasks and results. Unlike MHPCs of Section 2.1, MFs have no computational capacity and can only store any message and transfer it between two entities. Thus the fundamental difference between the MHPC work of Chapter 2 and the HPC+MF work of this chapter is that while in the former the computation and communication resources of the system are both combined in the MHPC, here the computation resource is the HPC and the communication resource is MF. We call this feature that is present in the MHPC and non-present in the problem presented in this chapter, *Computation on the Move* and discuss its possible advantages and disadvantages in Chapter 4 in more detail.

Our presentation of this problem, named the HPC+MF problem, similarly to Section 2.1, focuses on the controlling of the mobility of these communication resources (MFs) as well as proper scheduling of computation on the computation resource (HPC) in order to achieve various objectives including finishing computation as soon

as possible, minimizing travel and fuel consumption, providing the most timely service to all users in the area with computationally intensive tasks, etc. Complementing our work is the body of literature on Cloudlet seeding [46]. This work is similar to that of this chapter in the sense that it also suggests providing computation resources via a stationary, more powerful computer, but it differs from our work as Shires et al. focus on placement of these resources, called cloudlets in their work, in multiple locations each within reach of multiple user nodes via a direct short-range communication channel. Questions of where and how many cloudlets to place are addressed in that line of research. In contrast, our work assumes that a single HPC is placed at a location and we focus on making it reachable through MFs; hence questions of how to move the MFs are more relevant to our work.

Similar to the MHPC problem of Chapter 2, this problem finds its applications in situations where connectivity to the infrastructure Internet is not readily available and also when the area of coverage is large relative to the reach of communication channels that can support a reasonable bandwidth for communication of the tasks. In the environments that we consider, one feasible solution is utilization of a mobility-controlled vehicle (MF) to serve the computational tasks of all users in the area through a stationary computation resource (HPC).

To illustrate the problem setting, consider the simple system shown in Figure 19. An MF travels in the region at an average speed of 10 m/s. Tasks are served in a stationary HPC with a single preemptive processor[1]. Assuming that we describe distances as driving time of the MF, user Node 1 is located five minutes away from the current location of the MF, while User Node two is located two minutes away and the HPC is located slightly less than six minutes away. The user nodes are four minutes apart from each other and the HPC is nine and five minutes away from them respectively. In this simple example, the user nodes are stationary for the time period

[1]For a formal definition of preemptive processing, refer to Section 2.2.4

under consideration. Each user node generates a task at time 0. Node 1 has a task that will take 10 minutes on the HPC, while Node 2 has a task that will take three minutes. Assume the HPC processor allows preemption, thus a task that is started on the processor can be interrupted to allow work on another task, with resumption later from the point of interruption. The MF needs to pickup and drop off each task at the HPC before it starts processing.

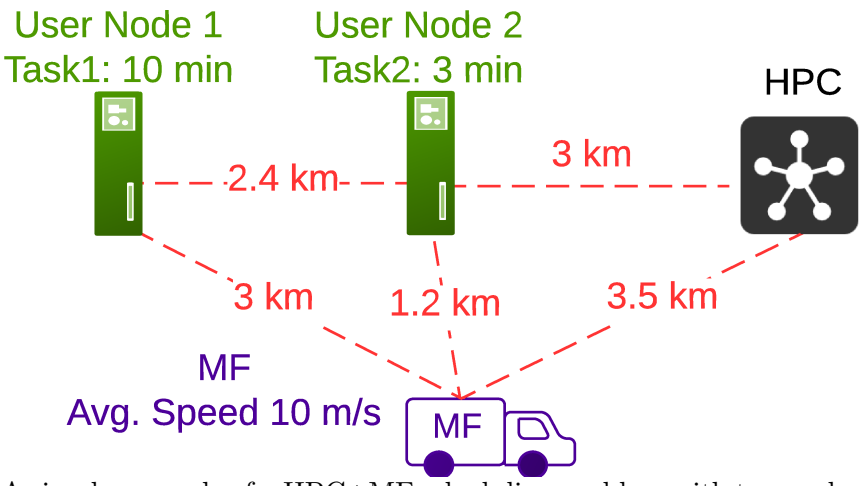


Figure 19: A simple example of a HPC+MF scheduling problem with two nodes and a single processor HPC. Sample task and MF features and distances among entities are shown on the figure.

In what order should the MF visit the task locations? In what order should it drop the tasks at the HPC and in what order should it receive the results for delivery from the HPC? In what order should the HPC processor work on the two tasks? What is a reasonable metric for quality of a solution? What changes if task 2 is not known to the HPC until two minutes into the scenario? Our HPC+MF framework and solutions address solutions to such problems.

We describe a formal framework for the above mentioned problem is Section 3.2. We then propose a mathematical formulation of this problem is Section 3.4 and use this formulation to derive some insights and develop scalable heuristics for a version of the problem that assumes complete in-advance knowledge in Section 3.5. In Section 3.6, we extend the previous heuristics to deal with more realistic scenarios where

information is revealed as users request computational assistance from the HPC. Finally, we present evaluations in Section 3.7 that further our knowledge of how the system is affected by various parameters and how to make decisions about choosing resources when serving a given scenario.

3.2 Problem Framework

3.2.1 Framework Structure and Problem Settings

In this section, we describe the general framework in which the HPC+MF problem is studied. We consider a system with mobile user nodes traveling in a bounded geographic area. Also present in the area are V MFs, initially located at known locations. There is single HPC in the area that is equipped with m processors; i.e. it is capable of executing m computational tasks simultaneously, one per processor. MFs are assumed to be identical and to take a known constant time to travel between a pair of nodes as a function of locations of those two nodes. Finally an MF Controller unit is present in the area that coordinates the efforts of the MFs to pickup and deliver the tasks/results. We assume that this controller is a part of the HPC unit and hence we refer to both the controller and the processing unit as HPC hereafter. This will be discussed in more detail in Section 3.3. We introduce the problem with these simplifying assumptions to streamline the exposition and allow the key insights to be highlighted. A high-level view of system framework is shown in Figure 20.

We assume that user nodes notify the HPC of the existence of computational tasks for offloading using a long-range, low-bandwidth radio such as the ones suggested in [57, 9]. This radio is assumed to be only useful for control plane operations and not capable of exchanging tasks/results. The user nodes are mobile, thus to facilitate task and result exchanges in the future it is necessary to designate meetup opportunities. In our formulation, the user node specifies a location where the MF can pick up the task and a location (possibly different) where the MF can deliver the result. Without

loss of generality, we have demonstrated examples of these locations in Section 2.2.

The HPC unit also is intended to represent a base station with abundant power supply and processing power. In military settings, the base stations [48] are usually equipped with satellite that can act as a gateway to the Internet plus high-performance local computer with access to generators that eliminate power constraints. In other scenarios, e.g. disaster scenarios, the base station can represent a high-performance computing power set up in a fixed location.

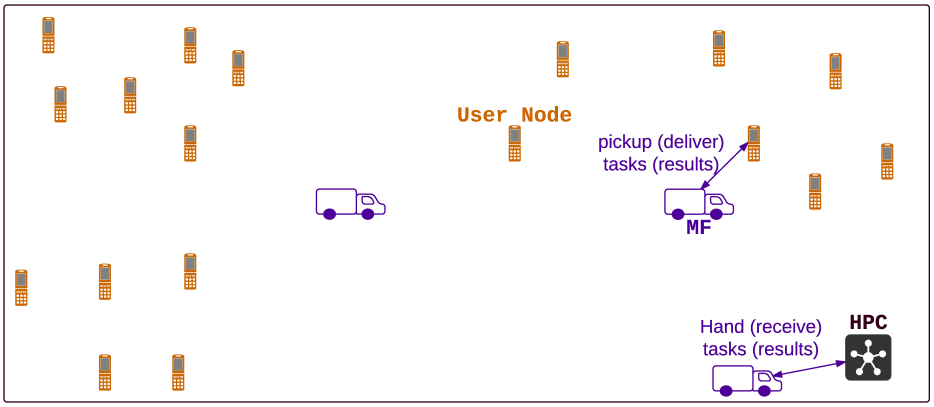


Figure 20: A framework for the HPC+MF problem. The *HPC* is notified about the tasks at the *User Nodes*. These tasks are picked up by *MFs* and dropped off at the HPC for processing and given back to the MFs for delivery to the user nodes.

As far as the MFs and the HPC are concerned, tasks arrive in groups of one or more generated at different locations. In a general case, this can imply some batching of the tasks before they are picked up by the MF and brought to the HPC for processing. As we will describe in Section 3.6, it may save some re-calculations of the solution if arriving tasks are revealed in groups rather than individually. Similar to Section 2.2, each task, i, in a group is represented by the vector $(R_i, D_i, T_i, LP, LD_i)$, where R_i is the time that task i is *available* for pick up at the location, LP, which is the same value for all tasks in the group, which denotes the time taken for the user node to get to the designated location. D_i is the deadline for the task, after which the user node is no longer interested in the result of the processing, and LD_i is the designated delivery location for the results of the processed task. In the case that $LP = LD_i \; \forall i$, we call

them *task locations* ;otherwise we call them pick up locations or delivery locations based on context. Finally, T_i is the duration of time that it takes to process the task on one of the processors of an HPC.

After being notified of the task, the HPC can commit an MF to pick up that task. Each MF initially plans to visit a number of task locations, pick up some tasks and bring them to the HPC. Once the HPC receives some tasks, it starts processing them. The details of this processing will be described in the algorithms of Section 3.5 and 3.6. Once a task has finished processing the HPC commits a MF to receive the result from the HPC and deliver it to the desired user node. The HPC may decide to wait for a number of tasks to finish processing and then give the results of them together to a MF. It may also commit the same MF to pick up some other tasks that have been revealed to the HPC from the last visit of a MF. It is also notable that there is no requirement that the same MF that has picked up a task must be responsible for the delivery of the result. The assignments of MFs for pickup and delivery of the tasks are made by the HPC in order to provide better service to the system according to some objective value (see Section 3.2.3 for more information). This is in contrast to the MHPC framework of Section 2.2 where an MHPC is committed to pick up, processing and delivery of the same task and no inter-MHPC communication is allowed. Another difference of the HPC+MF problem compared with the MHPC is evident in the fact that while in both problems each task location is only visited once for pick up and once for delivery[2], in the HPC+MF problem the HPC location can be visited many times. This means that based on the objective value requirements the HPC might dictate to the MFs to visit its location various times for dropping of their picked up tasks and receiving the processed results.

Deadlines D_i of the tasks can also be interpreted similarly as an estimate of the

[2]Pick up and delivery location of a task can be distinct in both problems. In this case, there will be one visit to the pick up and one visit to the delivery location per task.

local running time of the task. In this case, the rejection of the task by the HPC means that the user node will obtain the result faster if it processes it locally considering the requirement of two MF travels between the node location and the HPC location. Further work on offloading decisions can be found in [13]. Another interpretation of a deadline is an estimate of the remaining power on the user's hand-held device. In this case, delivery of the result will not be helpful, since the device's battery will be depleted before the result is obtained.

Note that there is significant flexibility in our task arrival and availability formulation. For example, we can model the scenario where all tasks are immediately ready by setting all availabilities of tasks to the corresponding times when they are generated. Conversely, we can relax the deadlines by setting them to a sufficiently large value.

An MF can *pick up* a task if it visits the pick up location of the task at or after the task availability time. The MF travels in the region, picking up available tasks and dropping them off at the HPC location. The HPC then plans for scheduling these tasks on its processors. As it will be described in Section 3.6, the arrival of groups of tasks affects the MFs' mobility schedule and the HPC's processing schedule for new pickups, deliveries, and processing, but does not affect its previous commitments. Note that there is no recommendation about picking up all available tasks first, processing them at the HPC, and then delivering all the results, or dropping off a single task to the HPC before getting to another nor there is any recommendation on how many times an MF responsible for a number of tasks shall visit the HPC. All such decisions are made only to achieve the minimal objective value.

In this new framework, pick up and delivery of tasks from its location and drop off and receiving of the task/result at the HPC location is done similarly to the MHPCs of Section 2.2. They are either done through a wired connection in case of user nodes exchanging tasks through FOBs or using a short-range, high-bandwidth radio in the

case of a designated location. We assume that the transmission range for this radio is negligible compared to the dimensions of the field where MFs travel. if that is not the case, instead of using MFs serving the HPC, a stationary HPC, similar to the cloudlets of [46], can be placed in a proper location where it can communicate with all user nodes. We assume that any time required for task pick up and result delivery as well as task drop off and receipt of results is negligible compared to travel and execution time. This assumption can also be lifted by adding a communication delay proportional to the size of the task / result for each data exchange.

We also assume that the HPC execution schedule is computed in a bounded amount of time, given the current task awaiting for processing at the HPC and the newly picked up tasks on their way to the HPC. This time is assumed to be negligible compared to the duration of the tasks. Finally, we assume that each MF can store all received tasks pending their execution and can store all execution results pending delivery; i.e. the size of the buffers at the MFs is much larger than the amount of information that are required to hold at a time.

3.2.2 Structure of the Solution

We are interested in producing a schedule for our HPC+MF system that has three components:

- **MF Assignment:** This specifies the subset of tasks that are assigned to each MF for pick up or delivery. Each MF will be responsible only for the specific service on its assigned tasks. No inter-MF communication is allowed in the model. Note that a task can be picked up and dropped off at the HPC by one MF while the corresponding processed results can be received and delivered by another MF. The assignment of these services to the MFs are implied in the MF Tour explained below.

- **MF Tour:** This is a pick up / delivery / HPC location visit schedule for each

MF and comprises a vector of tuples in the form (*location id*, *location visit time*, *other information*). There are two classes of locations that an MF will visit. One is the location of a regular node which will be visited for task pick up or result delivery. The other is the location of the HPC which will be to drop off tasks that the MF has picked up and possibly receive some of the results that the HPC commits the MF to deliver.

For a regular node, the generic visit entry above is in the form (*location id*, *location visit time*). In this notation, the location visit time specifies the time that the MF arrives for pick up / delivery at a regular node. For example, an entry $(\pm i, r_{\pm i})$ specifies that the MF arrives at location i for pick up $(+)$ or delivery $(-)$ at time $r_{\pm i}$.

For an HPC, the generic visit entry above is in the form (0, *HPC visit time*, *HPC visit duration*, *Pickup list*, *Delivery List*). In this notation, 0 stands for the location id of the HPC. HPC visit time time specifies the time that the MF arrives at the HPC for task/result exchange. HPC visit duration refers to the time the MF may have to wait, in the case when the HPC commits it to delivery of some results that are not yet ready and hence MF waits till they are ready to be received from the HPC for delivery. Pickup list is the list of tasks that the MF has picked up since its last visit to the HPC and is dropping off for processing. Delivery list is the list of tasks that have been processed by the HPC and are handed to the MF for delivery.

- **Task Execution Schedule:** This is the schedule of task execution on the processors of the HPC. This component comprises m vectors, one for each of the m available processors of the HPC. Each vector consists of tuples in the form (*interval*, *scheduled task*), where *interval* denotes the time period when the HPC is working on the specified *scheduled task*. For example, an entry

$((t_1, t_2), i)$ for processor j specifies that processor j is working on task i from time t_1 to t_2, inclusive.

3.2.3 Objective Value

In this section, we describe the various objective values that can be minimized as our goal in the HPC+MF problem. Similar to the MHPC problem, the objective value of the problem is an independent module that can be replaced based on the specific requirements of the instance of the problem to be solved. Any objective value that depends on travel times and task durations and other known inputs of the problem can be interchanged regardless of the details of the algorithms used to solve the problem.

A few possibilities for the objective value that we have considered are listed below. These objective values can be either directly integrated into the instance of the problem that is being solved or they can be calculated on a solution that has been provided for the problem to study the effect of minimizing one objective value on the value of the other. For example, we study the effects of providing a timely service on the amount of trips made by the MFs. Most of these objective values are very similar to those defined in Section 2.2.3 that have been re-defined for the new framework.

- **Return Time (RT):** This is one of the simplest and most natural objective values for the problem. RT is defined as the time that it takes for the latest MF delivers the processed result of the last task to which it has been committed to its designated location. This is used if total service time, in the view of the MFs and HPC, is of most interest.

- Time Throughput (TT): TT is similar to RT. It is defined as n/RT where n is the total number of tasks that the system has processed up until the current time. In essence, TT represents the number of tasks served per unit time that the system was running. One must note that TT shall be maximized for better

performance as a larger value means that more tasks are served in the same amount of time.

- **Average Completion (AC):** AC is defined as the average of the delivery times for all tasks by the MFs that have committed to them. This in turn implies the processing time of these tasks by the HPC as well. This is useful if providing early service to all user nodes is of most interest.

- **Average Flow (AF):** AF is the average of the difference between pick up and delivery for all tasks served by the MFs. Note that this metric is calculated in the view of each served task and the pick up and delivery MF are not necessarily the same. AF modifies AC by penalizing lateness in the delivery of the result only relative to the time that the task is revealed and avoids skewing to penalize tasks that are revealed later more than the earlier tasks.

- **Average Wait (AW):** AW is the average time that MFs wait at their visits to the HPC location before they can receive processed tasks for delivery. It must be emphasized that each MF might visit the HPC location numerous times. It is also notable that waiting can only occur at the HPC location and unlike the MHPC problem, a visit by an MF to a delivery location is simply dropping off a processed result that incurs no wait. This metric can be useful if vehicles need to have high mobility for strategic reasons.

- **Average Travel (AT):** AT is the average time that each MF is moving. This metric might be of interest in the cases that fuel consumption of MF is an important metric to be minimized.

- **Distance Throughput (DT)**: DT is similar to AT. It is defined as $1/AT$ or the total number of tasks that the system has processed up until the current time divided by the total distance traveled by all MFs. In essence, DT represents

the number of tasks served per unit distance that the MFs have traveled. One must note that DT shall be maximized for better performance as a larger value means that more tasks are served while traveling the same amount.

- **Normalized Average Flow(NAF):** NAF is defined below:

$$NAF = \max_{k \in Vehicles} \left(\underset{i \in Tasks}{\text{Average}} \frac{r_{-i} - R_i}{T_i} \right)$$

where, r_{-i} indicates the time when the results of the task are delivered. Other notations were introduced earlier in this section. This metric, like AF, envisions providing a timely service to each task that is requested relative to the time that the task is revealed, but it also normalizes this value by task duration so that larger tasks are not penalized for worse service simply due to their duration being longer than shorter tasks. Note that this metric will be equal to one if there is a dedicated MF that is present at the location of each task before it is revealed.

3.3 System Architecture

In this section, we describe the components of the system depicted in Figure 20 from a systems standpoint. The details of this section can help future researchers to implement their own version of our system.

A system deploying MFs that serve the HPC shares many of the same challenges faced by opportunistic communication systems. These challenges include concerns for accommodating various user node mobility patterns as well as using mechanisms for neighbor discovery that allow for efficient use of communication opportunities. In addition, the HPC+MF system, similar to systems using message ferries [10, 57], requires careful scheduling of ferry mobility as well as efficient coordination among the ferries when more than one is used [58]. The message ferrying with purpose, however, possesses significant additional features that warrant special attention. These

stem primarily from the fact that data delivery of MFs is always to/from the HPC. The availability of the results for the MFs to deliver also depends on the schedule of the HPC. Hence the message ferrying problem is interwoven with a scheduling problem. Thus the full communication requirements in this problem involves computation offloading to/from the HPC by the MFs and transfer of tasks/results to/from MFs by the regular user nodes that partly depend on the scheduling side of the problem. These features require new techniques for scheduling of movement of the MFs and scheduling of computation on the HPC. We develop an architecture for such an HPC+MF system in this section.

Figure 21 provides a high-level overview of the main components of the HPC+MF system. It consists of three components: a number of *User Nodes* that generate computational tasks and seek service from the HPC; a set of *MFs* which are vehicles with storage capability that are responsible for providing communication service; and an *HPC* that integrates an MF controller which manages the requests received, and dictates to MFs their future plan for picking up and delivering tasks. It also plans execution of these tasks on the processors of the HPC once they reach it. We also describe the *Radio Environment* through which the communications happen as a part of this architecture.

Radio Environment The three components can exchange high-level task metadata using a long-range, low-bandwidth radio such as those suggested in [57, 9]. Wide-coverage, low-bandwidth infrastructure using unlicensed bands may also be available and has been deployed [47]. This radio is assumed to be only useful for control plane operations and not capable of exchanging tasks/results. Actual tasks and results are exchanged via a short-range, high-bandwidth channel. Lewis, et. al [29] investigate a detailed client-server architecture along with the proper communication protocols for such exchanges of information.

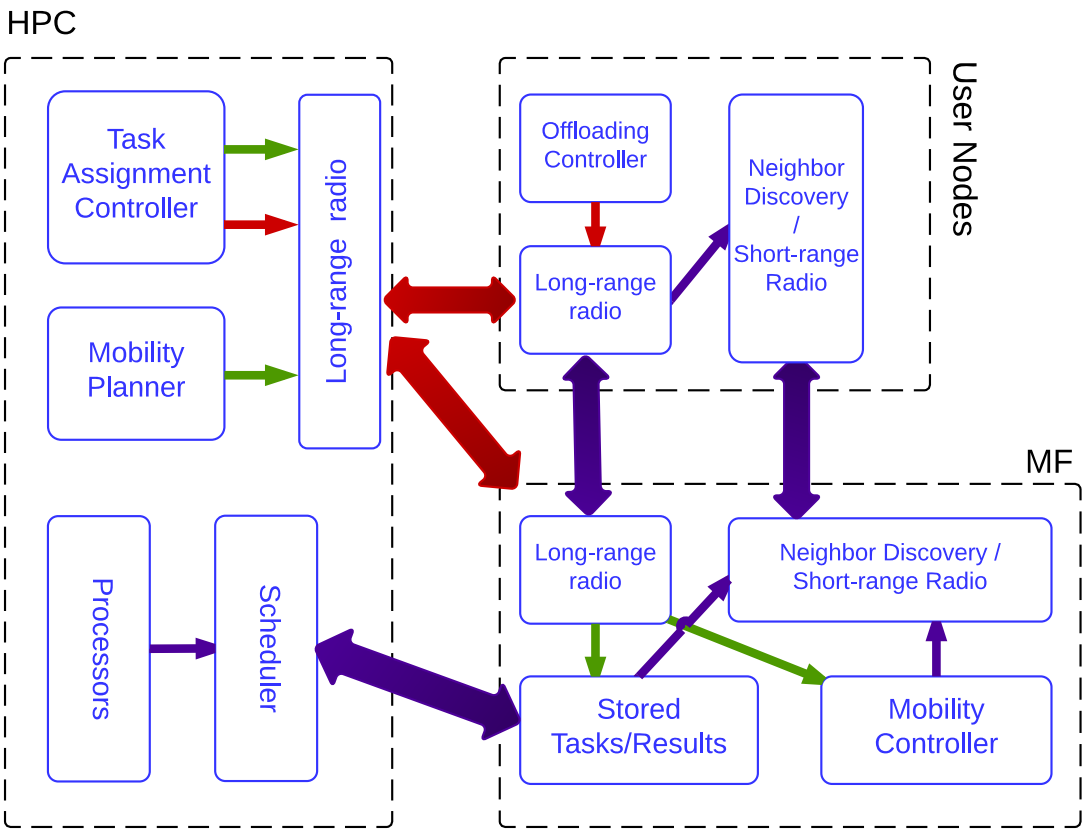


Figure 21: A system architecture for the HPC+MF problem. Three components of the system are: (1) *User Nodes* that own the tasks, (2) *MFs* that provide the communication service by picking up tasks, dropping them off to the HPC, receiving the results from the HPC, and delivering them back to the user nodes (3) *HPC* that provides computation service by processing the tasks. It also plans the mobility of the MFs.

User Nodes User Nodes are the owners of the tasks. Once a task is generated at a User Node, its **Offloading Controller** module decides whether the request must be sent with the task metadata to the HPC or the tasks must be processed locally. Various reasons ranging from privacy to time trade-offs may contribute to a decision to process the task locally, the details of which are outside the scope of this work. The interested reader can refer to [45] for examples of work focusing on offloading decisions. Once the User Node sends out the task metadata, if it does not receive a rejection via the **Long-range Radio** from the HPC, it will continue communicating with the assigned MF if necessary until the MF is within its range so that the task can be picked up via the **Short-range Radio** module. If an MF is assigned to

deliver results to a User Node by the HPC's Mobility Planner, the User Node will be contacted by the assigned MF and they will continue communicating until the MF is within its range so that the task can be picked up via the Short-range Radio module.

MF The MFs are the providers of communication service in the system. Multiple MFs can operate in the same area. In addition, each MF can provide communication service to multiple tasks simultaneously. The MFs receive their assigned tasks for pick up and delivery through a mobility plan from the controller embedded in the HPC via the long-range radio. They report on the state of their allocated pick up / delivery to the Controller over the same radio. The **Mobility Controller** supervises the mobility of the MF to follow the tour indicated by the **Mobility Planner** of the HPC to visit the User Nodes for pick up and delivery or to visit the HPC for drop-off and receipt of tasks/results. If the User Nodes are mobile, this module needs to use the long-range radio to repeatedly query the User Nodes for their current location and plan accordingly to meet them. Once the MF is in range with a User Node as notified by its **Neighbor Discovery** module, it can exchange the corresponding tasks/results to the User Nodes.

HPC This component handles the task offloading requests received from User Nodes. These requests are in the form of task metadata using a long-range and low-bandwidth radio. The HPC uses knowledge of MF and User Node locations as well their states to compute a *mobility schedule* for the MFs and a *task execution schedule* for its own processors. The **Processors** of the HPC work on the picked up tasks according to this specified execution schedule. The components of these are detailed in Section 3.5 and 3.6. The mobility schedule is communicated to the MFs and updated as needed via the long-range radio.

In this thesis, our main focus is to propose mechanisms for the HPC unit and how it schedules mobility and computation of the units to provide the best service to the

user nodes.

3.4 Mathematical Model of HPC+MF Problem

In this section, we propose a mathematical model to describe the message ferrying with purpose problem. To model the this problem, we need to make further simplifying assumptions to the general problem stated in Section 3.2. We consider complete knowledge of the tasks into the future. We also restrict this formulation to have a single HPC with a single non-preemptive processor serving tasks $1, \ldots, n$. We allow for multiple MFs in this formulation. Further, we assume that all tasks are available to be processed from time zero and there are no deadlines. We have intended the model in this section to present a slightly more complex version of problem compared to that of Section 2.4 to capture more variables in the problem.

Tables 2a and 2b summarize the parameters and decision variables used in the following formulation of the problem for the above framework.

Table 2: Parameters and Decision Variables used in the HPC+MF formulation.

(a) Summary of known parameters in the problem formulation.

Param.	Interpretation
$+i$	Id for pick up of task i
$-i$	Id for delivery of task i
T_i	Time duration of task i
t_{ij}	Travel time between location of tasks i and j
M	A very large number

(b) Summary decision variables in the problem formulation.

Decision Var.	Interpretation
X^r_{ijk}	1, if task j is visited right after i by vehicle k in its r^{th} trip and 0 otherwise
t^r_{0k}	Time that vehicle k reaches to the HPC location in its r^{th} trip.
w^r_{0k}	Waiting time of vehicle k at the HPC in its r^{th} trip.
s_i	Time that task i starts processing
y_{ij}	1, if the task j is scheduled right after i and 0 otherwise

In this model, we represent the HPC as node 0 and the tasks as 1 through n. For the purpose of keeping track of the variables and for each task i, we create a

pickup version $+i$ and a delivery version $-i$. We use the set $\mathcal{N} = \{\pm 1, \pm 2, \ldots, \pm n\}$ to represent all pickup and delivery versions of the tasks, and the set $\mathcal{N}^+ = \mathcal{N} \cup \{0\}$ to include the HPC as well. .This notation allows us to reduce the problem to a combination of a Multiple-Trip Vehicle Routing Problem (MTVRP) given some precedence and scheduling constraints and a Processor Scheduling Problem.

These constraints can be described as follows:

- Each MF can visit the HPC location as many times as the solution requires it to. For all other tasks (pickup/delivery version), they can only be visited once per trip of each MF.

- Between pick up and delivery of each task, there has to be at least one visit paid to the HPC. This allows the task to be processed.

- Delivery of a task $(-i)$ can happen no sooner than the time it is picked up $(+i)$ plus the time required to process it (T_i). This means that a task can only be delivered if it has already been picked up and processed.

- Processing of task i must start sometime after it has been picked up and the HPC must be visited afterwards.

- In any visit to the HPC, if the task i is planned for delivery in the trip immediately following the current visit to HPC, the MF needs to wait at the HPC until this task has finished processing. This implies that any visit to HPC might incur wait times until all tasks to be delivered in the next visit are finished.

- If task j is scheduled right after task i and all the above constraints are satisfied in such scheduling, it must start at a time later than the time task i has started processing plus the duration of the task, T_i. Note that task j starts either immediately after task i if it has already been picked up, or it starts once the task has been picked up.

This problem can be formulated as a Mixed-integer Linear program (MILP) as follows:

$$\underset{X,w,s,y}{\text{minimize}} \quad \underset{k,r}{max}\ (t^r_{0k}) \tag{16}$$

$$\text{subject to} \quad \sum_{i\in\mathcal{N}}\sum_{k=1}^{V}\sum_{r=1}^{R} X^r_{ijk} = 1 \quad ;\forall j \in \mathcal{N} \tag{17}$$

$$\sum_{j\in\mathcal{N}}\sum_{k=1}^{V}\sum_{r=1}^{R} X^r_{ijk} = 1 \quad ;\forall i \in \mathcal{N} \tag{18}$$

$$\sum_{i\in\mathcal{N}} X^r_{i0k} = \sum_{j\in\mathcal{N}} X^r_{j0k} \quad ;\forall k = 1,\ldots,V;\ \forall r = 1,\ldots,R \tag{19}$$

$$\sum_{i\in\mathcal{N}^+} X^r_{ihk} - \sum_{j\in\mathcal{N}^+} X^r_{hjk} = 0 \quad ;\forall k = 1,\ldots,V;\ \forall r = 1,\ldots,R \tag{20}$$

$$\sum_{j\in\mathcal{N}} X^r_{0jk} \geq \sum_{j\in\mathcal{N}} X^r_{0j(k+1)} \quad ;\forall k = 1,\ldots,V-1;\ \forall r = 1,\ldots,R \tag{21}$$

$$\sum_{j\in\mathcal{N}} X^r_{0jk} \geq \sum_{j\in\mathcal{N}} X^{r+1}_{0jk} \quad ;\forall k = 1,\ldots,V;\ \forall r = 1,\ldots,R-1 \tag{22}$$

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} X^r_{ijk} \leq |\mathcal{N}|-1 = 2n-1 \quad ;\forall k = 1,\ldots,V;\ \forall r = 1,\ldots,R \tag{23}$$

$$s_i \geq t^{r+1}_{0k} + M(\sum_{j\in\mathcal{N}} X^r_{ijk} - 1) \tag{24}$$

$$\forall i = +1,+2,\ldots,+n;\ \forall k = 1,\ldots,V;\ \forall r = 1,\ldots,R$$

$$s_i + T_i \leq t^r_{0k} + w^r_{0k} - M(\sum_{j\in\mathcal{N}} X^r_{-ijk} - 1) \tag{25}$$

$$\forall i = +1,+2,\ldots,+n;\ \forall k = 1,\ldots,V;\ \forall r = 1,\ldots,R$$

$$t^{r+1}_{0k} = t^r_{0k} + w^r_{0k} + (\sum_{i\in\mathcal{N}^+}\sum_{j\in\mathcal{N}^+} X^r_{ijk}t_{ij}) \tag{26}$$

$$\forall k = 1,\ldots,V;\ \forall r = 1,\ldots,R$$

$$s_i + T_i - (1-y_{ij})M \leq s_j \quad ;\forall i \in \mathcal{N}\ ,\ \forall i,j = +1,+2,\ldots,+n \tag{27}$$

$$t^0_{0k}, w^0_{0k} = 0 \quad ;\forall k = 1,\ldots,V \tag{28}$$

$$y_{ij} + y_{ji} = 1 \quad ;\forall i \in \mathcal{N}\ ,\ \forall i,j = +1,+2,\ldots,+n \tag{29}$$

$$X^r_{ijk} \in \{0,1\} \quad ;\forall i,j \in \mathcal{N}\ ,\ \forall k = 1,\ldots,V;\ \forall r = 1,\ldots,R \tag{30}$$

$$y_{ij} \in \{0,1\} \quad ;\forall i,j \in \mathcal{N}^+ \tag{31}$$

$$t^r_{0k}, w^r_{0k} \geq 0 \quad ;\forall k = 1,\ldots,V;\ \forall r = 1,\ldots,R \tag{32}$$

$$s_i \geq 0 \quad ;\forall i = +1,+2,\ldots,+n \tag{33}$$

In the above formulation:

- The objective (16) minimizes the latest time that one of the MFs returns to the HPC.
- Constraints (17) and (18) confirm that only one MF and only in one of its trips visits a task location for pickup or delivery.
- Constraint (19) confirms that while an HPC can be visited more than once by each MF, total in-flow and out-flow of HPC shall be the same.
- Constraint (20) confirms that the in-flow and out-flow of each pickup and delivery location of each task shall be the same.
- Constraint (21) confirms that a new vehicle among the V available vehicles is only used if a previous vehicle has already been used.
- Constraint (22) confirms that a vehicle makes a new trip only if in its previous trip at least one location has been already visited.
- Constraint (23) is a version of the traditional subtour elimination constraints used in Traveling Salesman Problem formulations.
- Constraint (24) confirms that the processing of a task can only start after the HPC is visited for the first time following the pickup of that task.

- Constraint (25) confirms that the delivery of a task can only happen if an HPC is visited for the last time (and at least once) before the delivery trip is started and after proper wait at the HPC before that trip is started.

- Constraint (26) indicates that the time between any two consecutive visits to the HPC by a MF is separated by the waiting time at the HPC after the former visit plus the time to travel among the locations to be visited in the trip.

- Constraint (27) indicates that if task i is processed before task j, the starting time of the processing of these tasks shall be separated by at least the duration of the task, T_i.

- Constraint (29) indicates that either task i is processed before task j or vice versa.

To solve this problem using this formulation a branch-and-cut or branch-and-bound method must be used, similar to those described in Section 2.4.

3.4.1 Complexity of the Heuristics

The HPC+MF problem is easily seen to be NP-Hard. A special case of the HPC+MF problem occurs when there is a single vehicle with a single processor and all the tasks are known in advance (A). A special case of (A) is MTVRP when the the minimum travel time among every location is larger than the maximum duration of the processing of the task. Since MTVRP is NP-Hard, (A) and HPC+MF are as well. It can be shown that even the simplified version of the HPC+MF problem presented above has $\frac{(3N)!(N!)}{2^N}$ feasible solutions. As a result, a brute force approach of trying all possibilities to find the solution is ineffective even for small number of nodes. This drives us to study the formulation and understand basic features of the problem that allow us to propose scalable heuristics for the problem.

3.5 Offline HPC+MF Problem

In this section, we develop heuristics for the offline version of the problem. Since the HPC+MF problem, as described in Section 3.4, is NP-Hard, we need to develop efficient algorithms to solve it. It is obvious that at the heart of the general problem described in Section 3.2 is an assignment of tasks to MFs given their commitments (similar to a bin-packing problem), a pick up / delivery order planning (similar to MTVRP) and a task scheduling problem that is solved on the processors of the single HPC. This encompasses the offline problem studied in this section. Since greedy algorithms are widely used to solve each of these problems, we adopt a similar approach to solve the offline HPC+MF problem.

To offer a scalable solution, first, we propose a greedy heuristic that builds a solution by considering one task at a time and assigning it to the best vehicle in the best tour location and task execution schedule assuming complete knowledge of tasks. In the next section, we extend this to non-complete knowledge.

3.5.1 Constructive Heuristic

The constructive heuristic builds the solution from scratch by inserting tasks one by one until all of them are placed. It uses a greedy approach to do so.

We describe a constructive heuristic that accounts for both preemption and non-preemption. The constructive heuristic starts with an empty MF Tour at all MFs and an empty Task Execution Schedule at the HPC (as described in Section 3.2.2) and then considers the tasks at user nodes one at a time and determines the best place in the current tour of each vehicle once for their pickup and once for their delivery. To select which vehicles and tour locations (or equivalently tasks) to visit at each step, the heuristic tries the current task under examination in each possible position in the tour of each vehicle for pick up and delivery (pick up and delivery MF tours can be distinct). This insertion shall be in accordance with the logical constraints

of Section 3.4. A solution needs to make sure that after a MF picks up each task, it visits the HPC location at least once to drop off the task. It also needs to make sure that before an MF delivers any result it has visited the HPC location to receive those results for delivery. There is also a requirement that the drop off visit of a HPC shall happen before the HPC visit for receiving the results of the same tasks. For each of these placements that does not violate the above requirements and the deadline requirements, it finds the best schedule. If the scheduler is preemptive, we use earliest delivery order first approach according to the order (or estimated time) when these tasks are supposed to be delivered. If the scheduler is non-preemptive, we use the simplified option of scheduling tasks with the same logic of earliest delivery first, except that preemptions are not allowed, even if a later task can improve the solution by preempting an earlier one. The offline heuristic for preemptive case is formally described in Algorithm 3.

The order in which tasks are visited for this greedy heuristic, noted in Algorithm 3, can be chosen in various ways including choosing tasks by index order, choosing the nearest (farthest) node to the MF, choosing the nearest (farthest) task to the latest currently inserted task. Alternatively, a "pure" greedy approach can be used that considers every task at every step and chooses the task that gives minimal increase in the objective value for final placement. This approach requires considerably more computation time.

3.5.2 Earliest Delivery Scheduling for HPC+MF

The preemptive scheduling in Algorithm 3 starts by assigning visit times to all tasks in the current tour purely based on travel times among locations and completely ignoring task durations. We call these "estimated visit times". It then schedules the task with the earliest "estimated delivery time". If this incurs a waiting time at the delivery HPC, the algorithm takes it into account and adjusts the visit times

of all locations in the tour accordingly. It then schedules the next task with earliest "estimated delivery" and repeats the same process until all tasks are scheduled. Since tasks are scheduled in their delivery order, preemption is implied and we do not need to explicitly enforce it, i.e. task that have priority are scheduled first and tasks with lesser priority are only taking the remaining processor time. This scheduling scheme is slightly more complex than that of Section 2.6 and is described in Algorithm 4.

3.5.3 Applying Heuristics to Solve Variants of HPC+MF Scheduling Problem

In this section, we discuss how the heuristics of Section 3.5.1, can be applied to the variants of HPC+MF Scheduling Problem.

To do this, we go over the list of HPC+MF Scheduling Problem variants again and describe the recommended choice of heuristics for each variant:

- **Multiple Processors:** The proposed heuristic accounts for the general case of multiple processors. A single processor problem is a special case of these with number of processors equal to 1.

- **Multiple Tasks per User Node:** With multiple tasks, all of the heuristics are applicable. It suffices to define a distinct pick up and delivery versions for each task of a location in that case. These location may coincide for which the heuristic is accommodating. Travel times among different pick up and delivery in the same location will be 0 in this case.

- **Objective Values:** All of the above heuristics build a tour greedily or improve it in the direction of a given objective. We have tested both TT and NAF as the objective value in our experiments. The addition of any other objective seems straightforward.

It must be emphasized that both heuristics are sub-optimal because they build a tour greedily and at each step they do not consider the future, or equivalently, it

does not consider that changing the past decisions that may result in an improved solution.

3.6 Online HPC+MF Problem

In this section, we turn into the online version of the HPC+MF problem. This is a generalization of the problem introduced in Section 3.5. In this case, we assume that knowledge of the tasks arriving is only revealed to the HPC and hence the corresponding MFs only after the User Node has requested service. We are interested in a *complete schedule*, defined as one in which all the tasks have been fully executed and their results delivered.

Before we develop a heuristic for this problem, we turn our interest to the proper choice of an objective value. One can keep using RT as an internal metric in this case for any sub-problem that uses Algorithm 3, but as an overall metric for the problem that considers incomplete knowledge of the tasks, RT loses meaning since a notion of "last task" loses meaning as well, i.e., the algorithm is not supposed to have knowledge of which task is the last, if any. Among metrics to measure the performance of the online problem, we have chosen NAF which is a metric that is not affected by tasks' arrival order. A closer look into the value of NAF defined in Section 3.2.3 shows that it considers the interest of user nodes in its numerator by trying to deliver the tasks as soon as possible while being fair by applying a normalization in the denominator to not prioritizing a long task over many short tasks[3].

3.6.1 Base Online Algorithm

In this section, we present the high-level view of the HPC+MF scheduling algorithm. The online instance is described above. To solve such online instance of the HPC+MF problem, it suffices that we repeatedly solve the offline problem any time a (group

[3]The algorithms described in this section and Section 3.5 are capable of replacing any objective value fitting other scenarios without affecting the behavior of the algorithm.

of) task(s) arrives. To see this, we introduce Algorithm 5 as follows:

The online algorithm uses the task information that is already revealed to obtain an initial solution. It then repeatedly "cuts" the solution (see Section 3.6.2) when new tasks arrive, recalculates the solution for the portion of the current solution that is not served after this revealing moment as well as the newly arrived information, and "merges" this new solution with the preserved portion of the solution (see Section 3.6.3). Note that the online algorithm assumes that the time taken to calculate the *New Solution* using Algorithm 5 is negligible compared to the time that it takes to process the tasks; otherwise, one can add this computation time as a function of the size of the problem and slightly modify the behavior of the HPC to address this. Our implementation allows for batching the groups of tasks in these cases. This batching will require the HPC to recalculate the solution only after either a specific amount of time has passed since the arrival of the first group of newly arriving tasks or a specific number of tasks have arrived. Batching is usually only needed to avoid frequent re-calculations of the solution. In our implementation batching is given and implied, i.e., we do not focus on the best practices to perform the batching and rather assume that revealing of tasks in groups rather than individual tasks is due to some batching that is underway.

3.6.2 Cutting Algorithm

As described in Algorithm 5, we need to cut the solution at the moment that a new group of tasks arrive. This action involves keeping the parts of the solution that have received partial service and re-planning for tasks that are not yet served or are partially served along with the new group of tasks.A task, in this framework, goes over the following phases that are depicted in Figure 22 as well:

1. It may not yet been picked up by an MF by the time that the new information arrives. In this case, the task is treated as another piece of new information

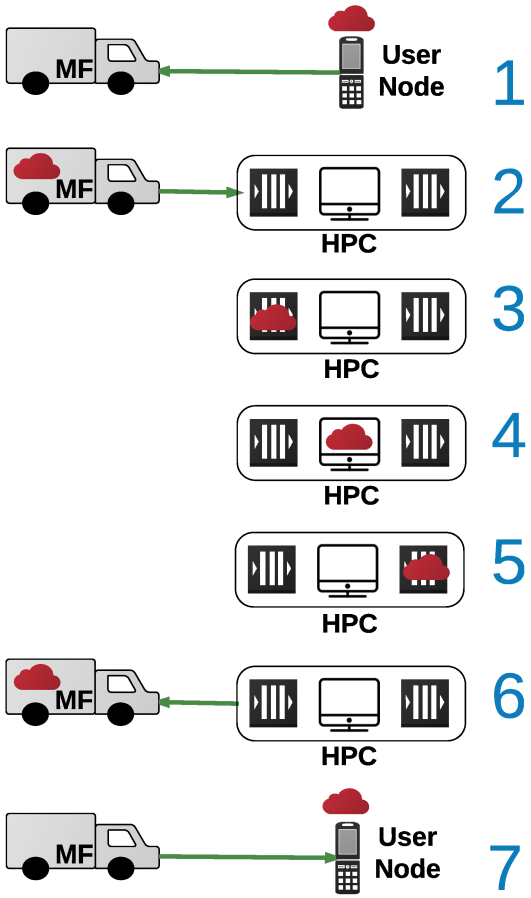


Figure 22: Life cycle of a task in the HPC+MF framework: (1) The task initially resides on user's hand-held device, (2) It is picked up by the MF and eventually reach the HPC, (3) It is received at the HPC but awaits processing, (4) It receives processing at the HPC, (5) It awaits to be transferred to an MF for delivery after completion of processing, (6) It is transferred to an MF that will eventually deliver it, (7) The result is delivered back to the user node.

that has arrived and must receive full service. This is phase 1 in Figure 22.

2. It may be picked up by an MF but has not yet arrived at the HPC and hence its processing may not have yet started by the time that the new information arrives. In this case, we keep that task on the pickup list of that MF, and while the MF might make detours due to arrival of the new information, it will drop this task off to the HPC once it visits the HPC location. This is phase 2 in Figure 22

3. It may be picked up by an MF and has arrived at the HPC, but its processing

may not have yet started by the time that the new information arrives. In this case, the HPC can continue planning for the tasks that it has awaiting processing, including this task. The delivery MF and delivery time may change due to re-planning for the newly arrived tasks though. This is phase 3 in Figure 22

4. It may be picked up by an MF, have arrived at the HPC and be partially processed by the time that the new information arrives. In this case, the HPC can continue planning for the tasks that it has awaiting processing, including this task. The HPC may choose to continue processing of this task or it may choose to preempt it for another task that in the new re-planing might need an earlier delivery. The delivery MF and delivery time may also change due to re-planning for the newly arrived tasks. This is phase 4 in Figure 22

5. It may be picked up by an MF and completely processed and waiting for a delivery service at the HPC. In this case, the HPC will assign an MF for delivering the results of this task. This assigned MF may have changed from the previous plan due to arrival of the new information. This is phase 5 in Figure 22

6. It may be picked up by an MF and completely processed and the results already transferred to an MF for final delivery. In this case, that MF will keep its commitment for delivery of the results of this task, but the time of delivery might change due to re-planning imposed by the HPC for the newly arrived information. This is phase 6 in Figure 22

7. It may be completely processed and delivered by the time that the new information arrives. In this case, the task does not need any further service and cutting has no effect on it. This is phase 7 in Figure 22

To explain the operations involved in cutting the solution at the moment t_i when a

new group of tasks arrives, assume, for the tour of k^{th} MF that that t_k is the moment that the MF finishes serving (i.e., delivering all the results) the last task among those for which it has been planning since the last period.

- If $t_i \geq t_k$, we assume that each MF has stopped at the location of delivery of its last task. When planning for a new group of tasks, we assume that these are the new initial locations and use Algorithm 5. All of the solution in the previous step is marked as *served* and Algorithm 3 solves the problem only for the newly arrived groups of tasks to deliver the *New Solution*.

- If $t_i < t_k$, we need to re-plan for any pick up, drop-off at HPC, processing, receipt of results from the HPC, and delivery that has occurred after t_i. To do this, we need to find the location that each MF will be at time t_k. At this moment, the MF will be either traveling between two user nodes, or a user node and the HPC. In both cases the MF location can be found with either of the following three possibilities: (i)using interpolation or (ii) the MF will be finished with the last task and its location will be the location of that task or (iii) it may be at the location of HPC without any current assignment. To cut the solution at t_k, we keep tasks of the first six categories described above and treat them accordingly for the completion of their service, whether it is pick up, dropping off at the HPC, processing, assigning the results to an MF, delivery, or a combination of those. We then add the new group of tasks to this list of *non-served* tasks and use Algorithm 3 to deliver the *New Solution*. Note that any of the tasks that remain after cutting the solution will be marked as *served*.

3.6.3 Merging Solutions

This operation merges the *New Solution* that results after applying Algorithm 3 with the *Base Solution*. This merging shall account for the offset the time that the new information is revealed, t_i, imposes. The merged solution will have the same format

as described for any solution in Section 3.2.2 and will be used as the Base Solution in the next iteration of the for loop in Algorithm 5. This will provide a complete solution to the problem.

3.7 Evaluation

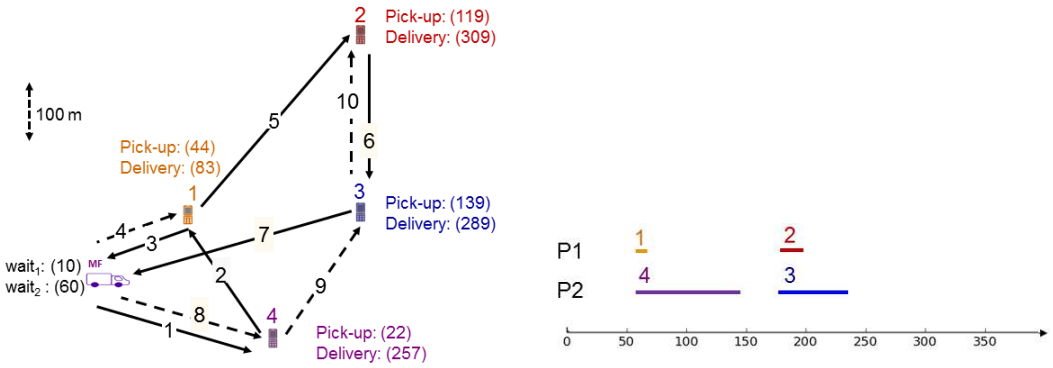
In this section, we perform various experiments to understand the performance of the algorithms presented in Sections 3.5 and 3.6. We start by reviewing examples of simple instances of the offline problem to understand the structure of the solutions. Then, we present various experiments that study the online heuristics that solve a more realistic instance of the problem.

3.7.1 Examples of the Offline HPC+MF Problem

In this section, we study some examples of performance of Algorithm 3 with preemptive scheduling. This gives us a preview of components of the solution, discussed in Section 3.2.2.

Our experimental setting for this section is depicted in Figure 23a. Assume that a two processor MF is initially at the location seen in the figure, and there are tasks of duration $10, 20, 60, 90$ sec at user nodes $1, 2, 3$, and 4 respectively, all known to the HPC at the beginning of time.

Since there is a single MF in the framework, we do not have any MF assignment in the provided solutions. Figures 23a and 23b show the MF Tour and Task Execution Schedule of the HPC for the instance described above using a preemptive scheduler in a pictorial manner. Pickup and delivery time of each task is shown at its location. HPC is visited twice and there is a wait of 10 and 20 sec respectively for the preparation of the results to be delivered by the MF during its next tour. It is evident from the solution that in this small instance, an ideal strategy is to pick up the two nearby tasks at locations 1 and 4 and wait at the HPC for the processing of the shorter task, 1, followed by delivery of the short task and pickup of all other tasks during the next



(a) MF Tour / preemptive Scheduler. **(b)** Task Execution Schedule.

Figure 23: MF Tour and Task Execution Schedule of HPC for a sample solution of an HPC+MF problem with a two processor HPC serving 4 user nodes. The MF Tour can be followed using the numbered arrows. Pickup and delivery time for each task is shown on its location. Each schedule shows intervals that either of the two processors of the HPC are busy processing a task.

round. The MF's final round includes a wait for all of these tasks to finish processing and delivery of all the results. This solution takes a total of 309 sec to serve all the tasks. To complete this example, we have shown the Task Execution Schedule component of the solutions in Figure 23b. Note that in this specific instance since there is not an occasion of the MF having more than two tasks awaiting processing, preemption is not exercised.

3.7.2 Performance of the Offline HPC+MF Heuristic versus Number of Processors for Non-preemptive and Preemptive Schedulers

In this section, we explore the effect of number of processors on RT as the objective value of examined instances to understand the advantages of preemption in multiple processor cases by comparing the preemptive and non-preemptive cases. In this experiment, we use assume 40 task locations that are mapped to locations of Georgia Tech bus stops in a $2km \times 2km$ area as shown in Figure 31. The HPC location is shown as a star shape on the Figure. 50 tasks are generated, each at one randomly chosen location among the 40 locations described above. We assume that pickup and delivery location of each task is the same. Durations of these tasks are derived from

the CERIT-SC workload log [3] which is a list of computational jobs submitted to the MetaCentrum distributed computing infrastructure[4] in the year 2013. Since some of these tasks were too short and a good number of them were too long, taking weeks to complete on the MetaCentrum infrastructure, we filtered the dataset to keep only tasks with durations between 60 *sec* and 3600 *sec*. We also assume all tasks are available at the beginning of time and no task has a deadline.

We have changed the number of processors from 1 to 10 and observe the return time (RT) time for both non-preemptive and preemptive heuristics. Figure 24 shows this effect for both experiments. We observe that as we increase the number of processors RT almost flattens for both types of schedulers when we have 8 processors. This happens as extra processors will not be needed for a fixed load in the system. It is also notable that preemptive scheduling results in a lower objective value which translates to better performance for the preemptive case. This is also intuitive as in general, non-preemption can be viewed as a special case of preemption that the algorithm can choose to adopt if it results in a better objective value. It is notable that the advantages of preemption are generally more evident with a lower number of processors.

3.7.3 Performance of the Online HPC+MF Heuristic versus Task Arrival Frequency

In this section, we study the effect of the frequency of task arrival on the system. We use a similar model to that of Section 3.7.2 for the basic environment settings that is described in below.

There are 3 MFs serving the system and a single HPC is placed as shown in Figure 31. we serve 600 batches of one or two tasks arriving according to a Poisson process. Pickup and delivery location of the tasks and their duration are chosen similar to Section 3.7.2. We vary the mean inter-arrival time for the exponential process that

[4] `http://metavo.metacentrum.cz/en/index.html`

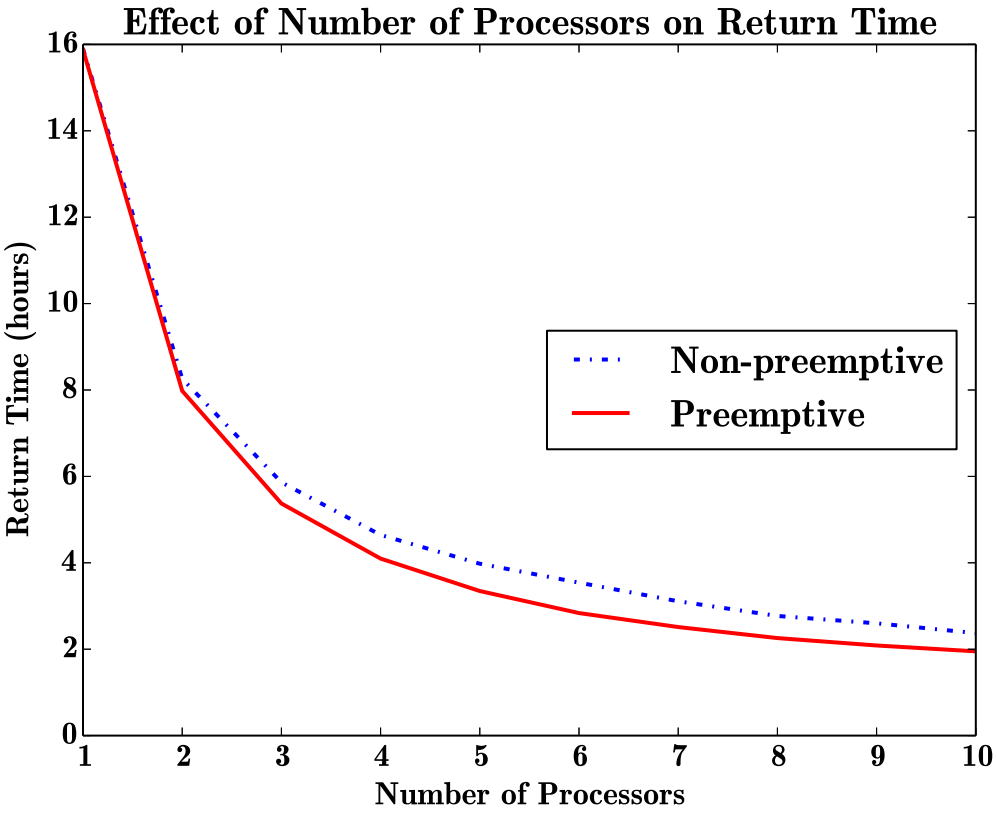


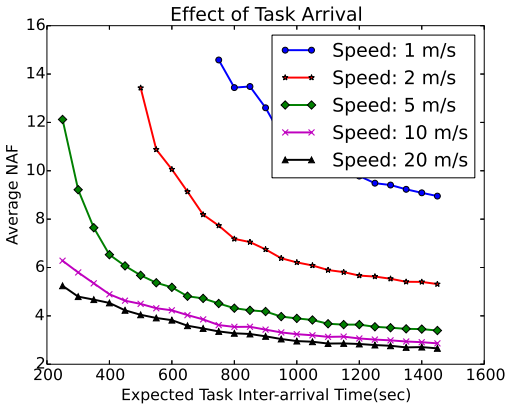
Figure 24: Comparison of the preemptive and non-preemptive scheduling for an HPC+MF problem. The x-axis shows behavior of each scheduler as the number of processors grow. The y-axis shows the objective value RT in hours.

governs the inter-arrivals for five sets of experiments with MFs traveling at speeds of 1, 2, 5, 10, 20 m/s. For each of these experiments, we repeat the experiment 20 times and average the results. In these exemptions, we have varied the expected inter-arrival time of tasks from a minimum value respecting the capacity of the system to 1500 seconds in 50 second steps[5].

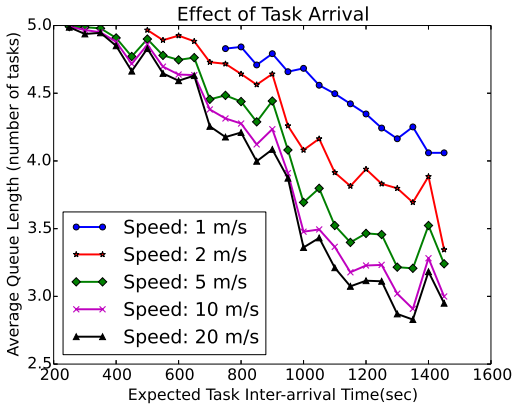
Figure 25a shows the average NAF (see Section 3.2.2 for definition) value for various speeds. It shows that both increasing the speed and increasing the inter-arrival time of tasks decreases the cost. However, in the latter once there is enough time to serve all the tasks that arrive before new tasks arrive, their behavior and

[5]The minimum arrival rate used for the speeds are one task every $750, 500, 250, 250$, and 250 seconds respectively for lower to higher speeds

thus the cost remains constant. To explore deeper into the effect of task arrivals, in Figure 25b, we have plotted *average queue length* for the same experiments. The average queue length is obtained by looking into the number of tasks that are awaiting processing at the HPC at discrete moments in time and averaging this number over the run. Figure 25b shows that in general average queue length drops with increasing the task inter-arrival time as well as speed. It is also notable that since each of the 20 runs of the experiment is a randomly generated instance, while the decreasing with larger inter-arrival time trend is true on the macro-level, it may not be true from one run to adjacent run due to randomizing effects. These figures collectively show that while the system experiences larger queues and worse performance on densely arriving tasks, after some threshold, its performance does not change considerably even with tasks arriving more sparsely.



(a) The Objective Value, average NAF.



(b) Average Queue Length for awaiting tasks at the HPC.

Figure 25: Effect of arrival frequency of tasks on the performance of online HPC+MF. X-axis shows the mean value for exponential process governing inter-arrival time of tasks. Tasks arrive more sparsely along this axis. The metric measured on Y-axis is noted o the corresponding captions.

3.7.4 Performance of the Online HPC+MF Heuristic versus Task Deadlines

In this section, we study the effect of deadlines. The purpose of this section is to understand the capacity of the heuristics of Section 3.6 in accommodating tasks with

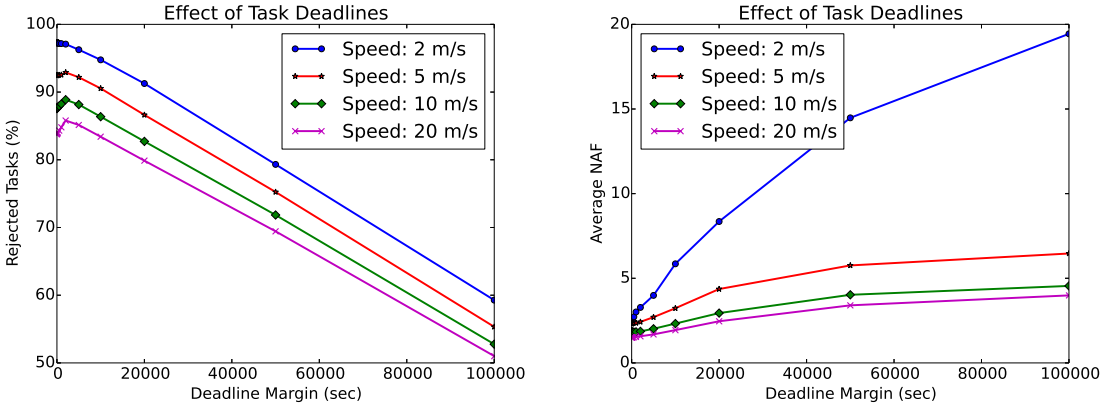
deadlines of various levels of tightness. We study the behavior of the heuristics from the standpoint of the percentage of the tasks rejected and the performance of the heuristics on the accepted tasks as the task deadlines become more relaxed.

Experiment setup for this section is similar to Section 3.7.3 with 3 MFs and one HPC having 2 processors. We serve 600 batches of one or two tasks arriving according to a Poisson process with an exponential mean inter-arrival time of 250 *sec*. These tasks have pickup and delivery location and duration chosen as described in Pickup and delivery location of the tasks and their duration are chosen similar to Section 3.7.2.

We vary the *deadline margin* to see how it affects task rejection, as well as cost. Deadlines for each task is defined as "task revealing time + task duration + deadline margin". Here, task revealing time is the moment that HPC is notified of the arrival of the new task. We use the values of 0, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000 *sec* as the value of deadline margin for five sets of experiments with MFs traveling at speeds of 2, 5, 10, 20 m/s. For each of these experiments, we repeat the experiment 20 times and average the results.

Figure 26a shows the percentage of all tasks submitted by user nodes that are denied service as the deadline margin grows larger and hence the service requirements become more relaxed. It is observed that with higher deadline margin there is a less portion of tasks rejected service. This can be easily seen since with loser deadline requirements , the HPC will have more time to process the tasks by the given deadline and the MFs are more likely to deliver the results before the deadline. As a reminder, note that the system makes estimate for when each piked up tasks can be delivered given the time that it can be picked up and processed and rejects the task for any service including pickup if it determines that the rest of the service cannot be completed in a timely manner to satisfy the deadline requirements. It is also observed that there is less rejection of tasks with higher MF speeds.

Figure 14b shows that initially the service given becomes worse with a higher deadline margin. This is seen when one considers that we a looser deadline requirement, there are more tasks accepted for service and the system is more overloaded providing a service of lesser quality to each task. Eventually as the deadline margin grows higher, we observe that the performance stabilizes. This shows that while for a very small number of tasks the system can provide a better service, there is a range of for the load for which the system is able to provide the same service. It is expected that if the load gets near system capacity one observes a sudden deterioration of service. The stabilization of service is more evident for higher speeds and not yet reached for lower speeds. This shows that the stabilization region is reached later if the MFs travel slower since such system has lesser capacity of service in general.



(a) Average percentage of rejected tasks. **(b)** The Objective Value, average NAF.

Figure 26: Effect of task deadlines on the performance of online HPC+MF. X-axis shows deadline margin of the tasks defined in Section 3.7.4. Deadlines become looser along this axis. The metric measured on Y-axis is noted o the corresponding captions.

3.7.5 Examination of Effects of the Relative HPC Location on the Online HPC+MF Heuristic

In this section, we examine the effect of the location of the single HPC in the framework of Section 3.2 on the performance of the online heuristic of Section 3.6. It must be emphasized that HPC placement is not the goal of the heuristics presented in this chapter. It is assumed in this work that the HPC is a fixed entity, e.g. a

military computing base station, that has access to an abundance of communication and computation resources and is placed in a location that is decided independent of the locations or mobility of the user nodes. The problem of HPC placement has been addressed in works like [46] for the case where numerous HPC cloudlets are placed in the area and groups of user nodes that are around each of them have access to the HPC via a direct communication channel. In our work, it is assumed that the HPC location is given and it is the job of the MFs to provide proper communication to the HPC for the user nodes. Hence, the results presented in this section are merely intended to understand and study the effect of the location of the HPC on the performance of the system and do not intend to propose placement heuristics for the HPC unit.

To test this effect, we use the framework presented in Figure 27. We place two User Nodes in locations U1 and U2 in a $5km \times 5km$ field. We generate 600 batches of one or two tasks where each task has both its pickup and delivery randomly chosen at location U1 or U2 and its arrival is governed by a Poisson process with a 300 *sec* exponential inter-arrival time between tasks. These tasks are served by a single MF. There is also one HPC in the framework that has two processors. We perform three sets of experiments with the location of the HPC at Loc 1(blue), Loc 2(red), and Loc 3(black) respectively. In each set of the experiments we perform three sub-experiments. One with all tasks of small duration (all being similarly 10 *sec* in duration), one with all tasks of medium duration (all being similarly 100 *sec* in duration), and one with all tasks of large duration (all being similarly 800 *sec* in duration). Each of the nine experiments described above is repeated 20 times and the results are averaged.

Figure 28a shows the average NAF for the described experiments. Our first observation of the results is that the difference in service is more obvious for short and medium tasks compared to large tasks. This is expected as for the case of very large

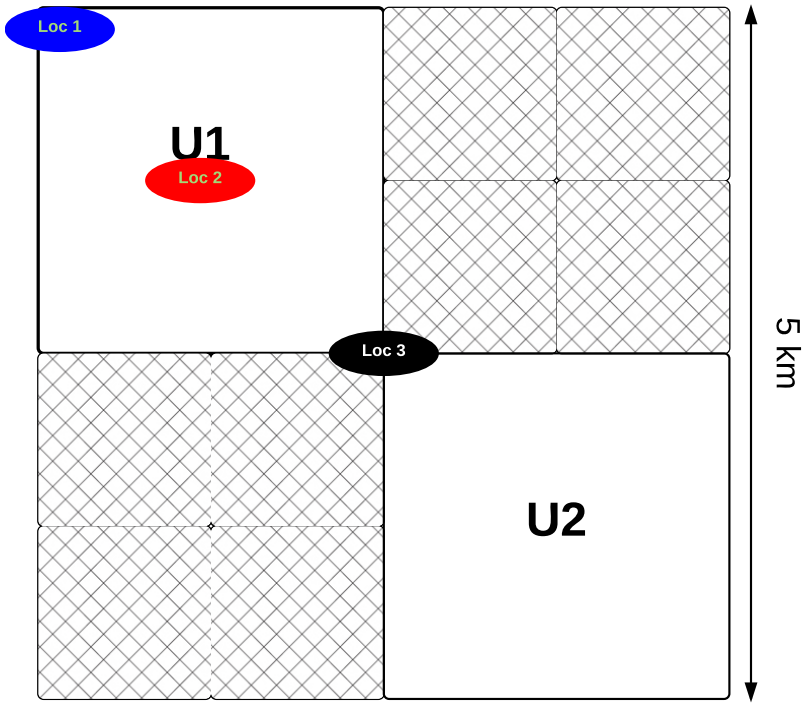


Figure 27: Framework for testing the effect of relative HPC location on the performance of the online HPC+MF heuristics. Two user nodes are placed at locations $U1$ and $U2$. The HPC is placed at $Loc.1$, $Loc.2$, and $Loc.3$ in various experiments.

tasks the processing and pickup/delivery order is less relevant as the service is dominated by the time spent to process the tasks at the HPC processors. Secondly, it is observed that for the short and medium tasks there is better service for the HPC being at location 3, location 2, and location 1 respectively. This shows that in general if HPC is closer to every user node, best performance is expected. Note that in case of location 2, the HPC is very near to one of the user nodes but far from the other one and in case of location 1 it is far from both. We expect that for more complex scenarios other features of the environment weigh in affecting the performance of the system.

Figure 28b shows the distance throughput for the same experiment. Since distance throughput only measures the amount of trips for serving each task, we do not see a distinction among the three cases of short, medium, and large tasks. It is also observed that for all three cases there is a higher throughput for the HPC being at

location 3, location 2, and location 1 respectively. This can be easily seen by noticing the fact that for the HPC being place at location 3, the average distance to both user nodes is smallest and for the HPC being at location 1 it is the largest. When the HPc is place at location 2, it is very near to one user node U1 and its distance to user node U2 is equal to the average distance to the user nodes. In contrast, when the HPC is at location 3, the average distance to user nodes is still the same as HPC being at location 2. Despite this fact, there is on average less travel made when the HPC is placed at location 3. This shows that when the HPC is generally more accessible for all user nodes, better decisions can be made for deciding when the MF travels to the HPC or to the user nodes and if some of these decisions are proven to be not optimal considering the future tasks, the effects of such mistakes are less evident.

3.8 Conclusions and Future Work

Motivated by message ferrying literature [57] and the idea of computational offloading, in this chapter, we expanded the architecture of a computational ferrying scheme of Chapter 2 to the case where the computation and communication resources are separated and Message Ferries (MFs) provide connectivity to a High Performance Computer (HPC) for a group of wireless user nodes. We studied the problem of scheduling the MFs' movement and HPC's processing. We started by modeling and formulating the design of such schedules as an optimization problem. We have discussed the challenges of this type of system and have developed algorithms that can be used to produce schedules for larger scale problems. We performed an extensive set of experiments to demonstrate applicability of the algorithms and show the effect of various system parameters on the performance of such this framework.

This work represents a first step towards the design and evaluation of the operation of an MF-based mobile cloud computing system. There are a number of interesting future directions which include:

- Understanding the robustness of an MF system to malfunctioning of its MFs and how to design failure recovery mechanisms by providing service through other MFs.

- Enhancements to the model to include variations such as processor-sharing scheduling on the HPC and the possibility of MFs picking multiple pieces of a computation from different locations or delivery to multiple user nodes.

- Integrating more complicated communication models in the framework, including modeling of wireless channels for both short-range and long-range radio with loss. This might require optimizing the heuristics of Section 3.5 and 3.6 so that the MFs starts exchanging information with the user nodes once they are at an optimal distance. There is also a requirement to deal with unsuccessful exchange of information due to channel imperfections.

- The consideration of communication among MFs either using user nodes as relays or direct exchange of information motivated by the work on multiple message ferry scheduling (e.g., [58]).

Algorithm 3: Offline HPC+MF heuristic pseudo-code

Input: Number of tasks: n, Specifications of each task i: $(R_i, D_i, T_i, LP_i, LD_i)$, Number of MFs: V, Number of HPC Processors: m.

Output: *Solution* = MF assignment (per instance): $\pm i_k \; \forall k \in V \; , \forall i \in tasks$, MF Tour (per MF): $tour_k = \{\ldots, (\pm i, r_{\pm i}, other\ information), \ldots\}; \forall k \in V$, Task Execution Schedule (per instance): $sched_k = \{((t_1, t_2), i)\} \; \forall k \in V$

```
1  Initialization: Mark all tasks as not-visited;
2  Sort the tasks to examine in some order;
3  while There exists not-visited tasks do
4  |  Take the next not-visited task, i.
5  |  for Each MF, K, in V do
6  |  |  for Each placement of task i's pick up in some MF's tour do
7  |  |  |  for Each placement of task i's delivery in some MF's tour (same or
   |  |  |  distinct MF from pick up) do
8  |  |  |  |  if If the above assignments do not violate the requiremenst of
   |  |  |  |  Section 3.4 then
9  |  |  |  |  |  Schedule task execution according to an earliest "estimated"
   |  |  |  |  |  delivery first and preempt existing tasks if necessary;
10 |  |  |  |  |  Store this assignment, tour, and execution schedule as
   |  |  |  |  |  Temporary Solution.
11 |  |  |  |  |  if Temporary Solution does not violate the deadlines and
   |  |  |  |  |  increases the objective value minimally then
12 |  |  |  |  |  |  Choose Temporary Solution as Solution.
13 |  |  |  |  |  end
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
17 |  end
18 end
19 Based on the Task Execution Schedule and MF Tour in Solution, calculate r_{±i}
   and w_i for each pick up / delivery. return Solution, Non-scheduled tasks as
   rejected.
```

Algorithm 4: Preemptive scheduling for HPC+MF based on earliest delivery

Input: Number of tasks: n, Specifications of each task i: $(R_i, D_i, T_i, LP_i, LD_i)$, Number of MFs: V, Number of HPC Processors: m, MF assignments: $\pm i_k \; \forall k \in V \; , \forall i \in tasks$, MF Tours: $tour_k = \{\dots, (\pm i, r_{\pm i}, other\ information), \dots\}; \forall k \in V$,

Output: *Preemptive Schedule* = Task Execution Schedule (per instance): $sched_k = \{((t_1, t_2), i)\} \; \forall k \in V$

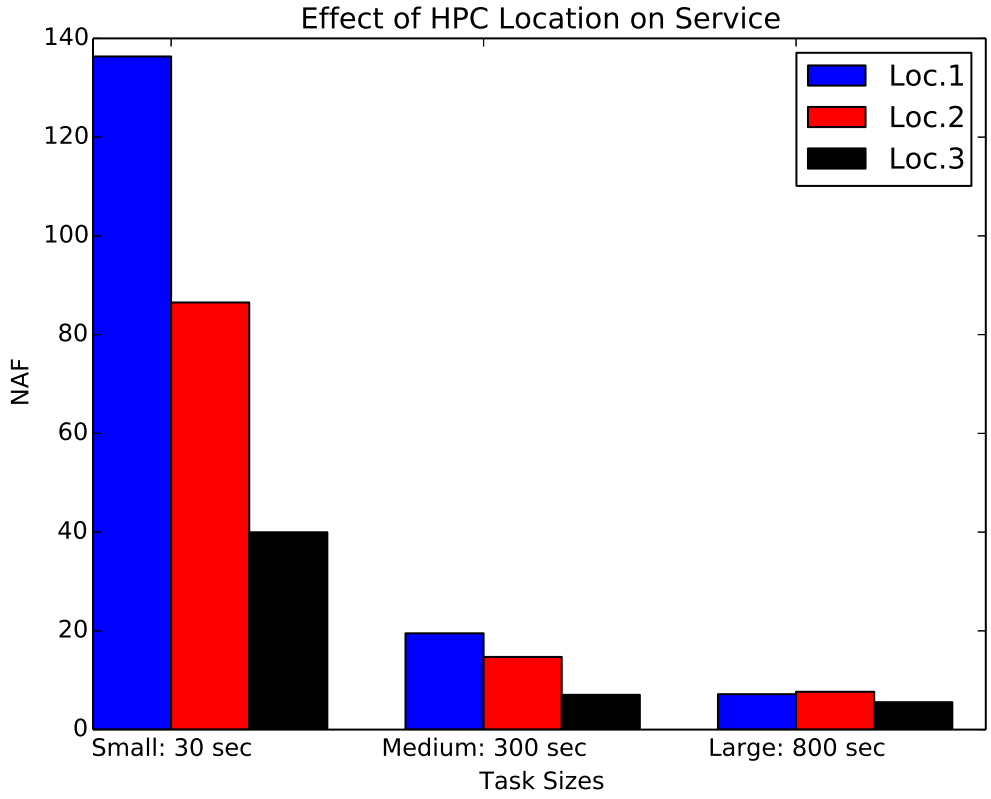
Initialization: Find travel time among all consecutive locations (ignore task durations). Store these values as *Estimated Visit Times and mark all tasks as not-scheduled*;
while *There exists not-scheduled tasks* **do**
    Take the task with earliest *estimated delivery time* (estimated visit time of delivery location). ;
    Schedule this task on any available (remaining processors). Processing can start as soon as task arrives at the HPC. Do not preempt earlier tasks. **if** *Insertion o the new Schedule incurs waiting at the delivery MF's visit to the HPC* **then**
        adjust the schedules and visit times according to the incurred wait.
    end
end
return *Final Schedule.*

Algorithm 5: Online HPC+MF heuristic pseudo-code.

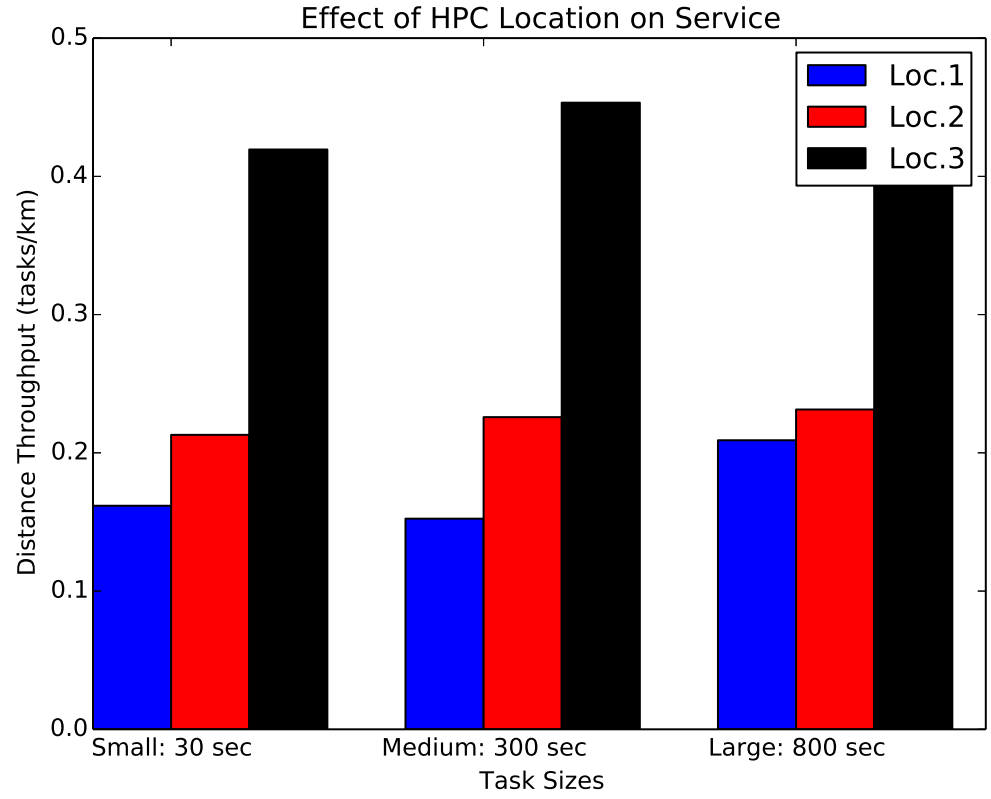
Input: Groups of tasks: (t_i, G_{t_i}), Number of MFs: V, Number of HPC Processors: m, Service period: t_S

Output: *Base Solution* = MF assignment (per instance): $\pm i_k \; \forall k \in V \; , \forall i \in tasks$, MF Tour (per MF): $tour_k = \{\dots, (\pm i, r_{\pm i}, other\ information), \dots\}; \forall k \in V$, Task Execution Schedule (per instance): $sched_k = \{((t_1, t_2), i)\} \; \forall k \in V$

Initialization: Use Algorithm 3 to solve for the tasks available at time 0 ;
Store this solution as *Base Solution.* ;
while *time* $< t_s$ **do**
    for *Each task group* (t_i, G_{t_i}) **do**
        Use Cutting Algorithm to cut the solution at t_i;
        Store the portion of the base solution before t_i and store it in *Base Solution.* ;
        Merge the portion of the solution after t_i with the tasks in G_{t_i} ;
        Use Algorithm 3 to solve for the merged input ;
        Store this solution as *New Solution* ;
        Properly Merge *New Solution* into *Base Solution*
    end
end
return *Base Solution* ;



(a) The Objective Value, average NAF.



(b) Average Distance Throughput in $tasks/km$.

Figure 28: Effect of HPC location on the performance of the online HPC+MF heuristics. Groups of bars on the X-axis shows the experiment for small, medium, and large tasks,

CHAPTER IV

TOWARDS UNDERSTANDING THE VALUE OF CONTROLLING MOBILITY IN A TACTICAL HIGH PERFORMANCE COMPUTING CLOUD SERVICE

4.1 Introduction

In this chapter, we present two simplified heuristics that intend to provide computational service in frameworks similar to those presented in Sections 2.2 and 3.2. To provide ground for these new heuristics, we note that in Chapter 2 we have used the concept of *Computation on the Move* to introduce the MHPC heuristics. In these heuristics the computation resource is mounted on the vehicles and hence the processing of a task can start as soon as the vehicle has picked it up. In Chapter 3, we have traded the Computation on the Move feature for having access to a possibly more powerful computation resource which is set up as a stationary HPC. The MHPC heuristics of Chapter 2 and HPC+MF heuristics of Chapter 3 both have one feature in common: they both control the mobility of the vehicles. Whether it is the MHPCs or the MFs, in both frameworks, the mobility of the vehicles is dictated through a controller which directs them where to go and in which order visit the task locations. This *Controlled Mobility*, allows for better optimization of service in the environment and aligns the efforts of the vehicles towards providing better service to the user nodes. In this chapter, we introduce heuristics that do not enjoy the Controlled Mobility feature.

Such scenarios can be reasonable in settings where the primary goal of vehicles is not serving the user nodes, but instead the vehicles follow a per-determined route and we utilize their mobility to provide service to the user nodes in the area. Examples of

these scenarios are road vehicles, e.g. buses. These vehicles follow known routes and one can envision either mounting computation resources (e.g. cloudlets) on them or using them as message ferries to serve computation tasks offloaded by user nodes that meet these vehicles on their path. A fundamental feature of this framework is that the vehicles are supposed to follow their pre-determined route and while the heuristic can optimize scheduling of computation, they will not have any control over mobility of these vehicles.

In COSMOS [44], Shi et al., suggest using their infrastructure for offloading computation to local resources and test the idea by using campus shuttles that experience variable connectivity to local wifi. Our work suggests the other side of this picture where the computation is stationary and resources can be mounted on the shuttle. In Cirrus Cloud [43], the authors classify computing resources as user-carried mobile devices, mobile computing resources attached to moving vehicles, infrastructure-based computing resources (similar to cloudlets), and Central cloud resources. They suggest utilizing mobility of vehicles like buses to provide communication among these elements that ultimately help the user-carried mobile devices to have access to better computation resources. FemtoClouds [23] also nicely categorizes the spectrum of using mobile entities as computing resources with respect to stability and predictability of their movement. Finally Zhao et al. [57] contrast two message ferrying schemes, namely, Node-Initiated MF (NIMF) and Ferry-Initiated Message Ferrying (FIMF). Both schemes support using message ferries to provide pure communication service in disruption tolerant settings. In contrast, our HPC+MF is comparable to FIMF that controls ferry mobility to provide computation service. A counterpart to their NIMF is an scheme that uses the predictable mobility of vehicles to transfer tasks and results to/from users and the HPC as they meet these vehicles along their routes.

Table 3 summarizes classification of our problems on the dimensions of Computation on the Move and Controlling Mobility and how the heuristics of Chapter 2 and

3 and this chapter are placed with respect to this classification.

Table 3: Classification of proposed heuristics according to the dimensions of "Computation on the Move" and "Controlled Mobility".

Comp. on the Move \ Controlled Mobility	Yes	No
Yes	MHPC Heuristics of Chapter 2	MHPC Heuristics of Section 4.2
No	HPC+MF Heuristics of Chapter 3	HPC+MF Heuristics of Section 4.3

4.2 Non-Controlled Mobility Service for the MHPC Framework

In this section, we describe the first of our two heuristics that do not control vehicle mobility. The framework for this heuristic is depicted in Figure 29. In this framework the MHPCs follow a repeated and pre-determined route. User Nodes can meet the MHPCs on their route. We assume that the time needed to exchange information between the MHPC and the User Nodes is negligible and exchange can happen once the user nodes are within range of the MHPCs on their route. One can also assume that the User Node locations in Figure 29 are meetup locations, similar to FOBs of Section 2.2 and User Nodes leave their tasks in these meetup locations so that the MHPCs can receive them.

In our proposed heuristic for the MHPC problem with non-controlled mobility, each MHPC has a known route. It keeps visiting locations on its route one by one. It is possible that a location is shared within the routes of two or more MHPCs. Once an MHPC visits a location with a User Node, it commits for processing all tasks at that location. It schedules each task in a First Come First Serve (FCFS) manner, i.e., there is no intelligence in the scheduling of the MHPCs. They simply schedule tasks for processing in the same order that they are picked up. When a task is picked up, it goes to a FCFS queue. If MHPC has an empty processor, the task

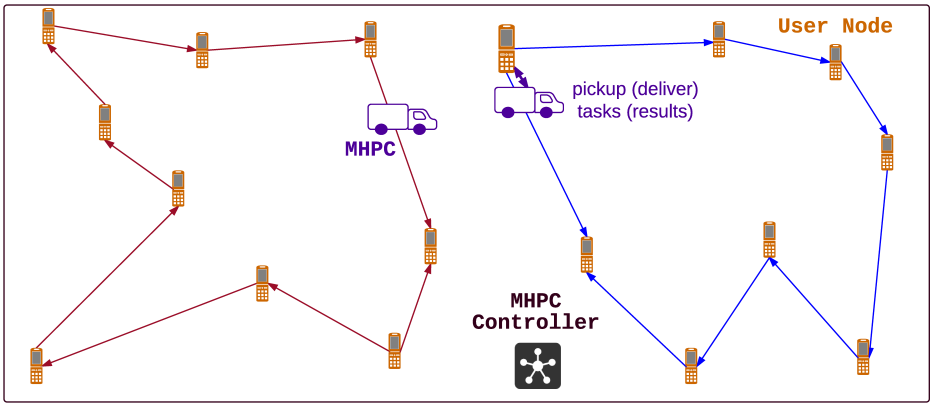


Figure 29: A framework for the MHPC problem without Controlled Mobility. The MHPCs follows a pre-determined route and User Nodes can exchange tasks and results as they meet the MHPCs enroute.

will immediately be scheduled; otherwise it will start processing as soon as all tasks picked up earlier than it have also started processing. It must be noted that if pickup and delivery location of a task are the same, for non-controlled mobility MHPC, this FCFS scheduler is equivalent to earliest delivery first scheduling of Section 2.6. Once a task is processed, it goes to a "ready queue" of the scheduler and is delivered once the MHPC visits the delivery location. It is assumed in this heuristic that either pickup and delivery location of each task are the same, or if they are distinct, they are both on the route of an MHPC; otherwise the task cannot be served properly under this scheme. This above heuristic is formally described in Algorithm 6.

4.3 Non-Controlled Mobility Service for the HPC+MF Framework

In this section, we describe the second of our two heuristics that do not control vehicle mobility for MFs and employs a stationary HPC. The framework for this heuristic is depicted in Figure 30. In this framework the MFs follow a repeated and pre-determined route. User Nodes can meet the MFs on their route. We assume that the time needed to exchange information between the MF and the User Nodes is negligible and exchange can happen once the user nodes are within range of the MFs

Algorithm 6: Heuristic for MHPC problem with non-cotrolled mobility

Input: Groups of tasks: (t_i, G_{t_i}), Number of MHPCs/processors per MHPC: (V, m), MHPC Routes: $\{L_i \; \forall i \in MHPC \; route\}$

Output: *Solution* = MHPC Assignment (per instance): $tasks_k \; \forall k \in V$, Task Execution Schedule (per MHPC): $sched_k = \{((t_1, t_2), i)\} \; \forall k \in V$

```
for Each MHPC k ∈ V do
    while True do
        Visit the next location, L_k, on MHPC route.
        for Each task i with pickup location LP_i == L_k do
            Assign task i to MHPC k ;
            Add j to the queue for MHPC's FCFS scheduler.
        end
        for Each task j that the MHPC's scheduler has finished processing do
            if Delivery location of j, LD_i == L_k then
                if Current Time ≤ Deadline for Task j, D_i then
                    Mark task j as served.
                else
                    Mark task j as rejected.
                end
            end
        end
        if all tasks are served an no further tasks are supposed to arrive to the
        system then
            Terminate
        end
    end
end
return Solution= {MHPC Assignment, FCFS Schedule for each MHPC},
Rejected tasks.
```

on their route. One can also assume that the User Node locations in Figure 30 are meetup locations, similar to FOBs of Section 2.2, and User Nodes leave their tasks in these meetup locations so that the MFs can receive them.

In our proposed heuristic for HPC+MF problem with non-controlled mobility, each MF has a known route. It is required that the MFs visit the HPC location on their route as the HPC is the processing unit in the framework (shown in Figure 30). Unless MFs can drop off tasks at the HPC location, no service can be provided in

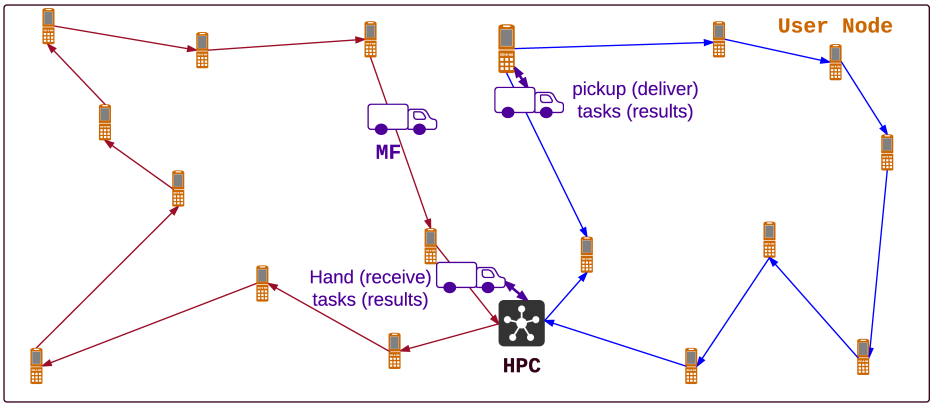


Figure 30: A framework for the HPC+MF problem without Controlled Mobility. The MFs follows a pre-determined route and User Nodes can exchange tasks and results as they meet the MFs en-route. All the MF routes need to meet the HPC.

the system[1]. It is possible that some other locations are also shared within routes of two or more MFs. The MF keeps visiting locations on its route one by one. Once an MF visits a location with a User Node, it picks up all of the tasks at that location. These tasks are stored on the MF until it reaches the HPC location. At that point, the MF drops all these tasks at the HPC for processing. The HPC schedules each task in a First Come First Serve (FCFS) manner, i.e., there is no intelligence in its scheduling. They simply schedule tasks for processing in the same order that they are picked up. When a task is dropped at the HPC, it goes to a FCFS queue. If it has an empty processor, the task will immediately be scheduled; otherwise it will start processing as soon as all tasks dropped off at the HPC earlier than the current one have also started processing. Once a task is processed, it goes to a "ready queue" of the scheduler. Every time that an MF visits the HPC location, the HPC hands all the finished tasks whose delivery locations are on the route of that MF. The MF then delivers these tasks once it visits their delivery location. This allows for one MF to pick up the task and another MF to deliver the result. This can happen under two scenarios. If the pick up and delivery location of a task are the same but are

[1]One can extend this framework so that each MF meets a distinct HPC on its route, or they can even meet a cloudlet that has Internet access. We assume that the MFs meet the same HPC to keep the framework comparable to the one presented in Section 3.2.

on the route of at least two MFs, or if the pick up and delivery location are distinct and each is on the route of a different MF. Note that in the non-controlled mobility MHPC heuristic of Section 4.2, in the former case, it is required that the same pickup MHPC provides processing and delivery service, even if the location is visited by other MHPCs and in the latter case service is not possible since the pickup MHPC cannot visit the delivery location and no inter-MHPC communication is allowed. This above heuristic is formally describe in Algorithm 7.

4.4 Evaluation

In this section, we start by describing the evaluation settings and then present various results that explore the value of controlling mobility.

4.4.1 Evaluation Setup

To create plausible scenarios for testing the heuristics described in this chapter, we have turned to a new testing platform.

In all of the below scenarios, we assume that the field of interest matches the map of Georgia Institute of Technology as show in Figure 31. This covers an area of approximately $2km \times 2km$. We generate tasks from a Poisson process at the locations of the bus stops, shown as black dots in Figure 31. We report the mean inter-arrival time of the resulting exponential process that governs arrival of the tasks to the controller in each section separately.

At each of the arrival moments generated from the above process, we randomly generate one or two tasks from randomly chosen locations among the bus stops. As it does not add any information to our experiments, we keep the pickup and delivery location of each task to be the same[2]. Arrival of one or two tasks in each arrival

[2]In general, all of the heuristics proposed in this thesis allow for distinct pickup and delivery location. We have only chosen these locations to coincide since their distinction did not reveal any additional observations for our experiments.

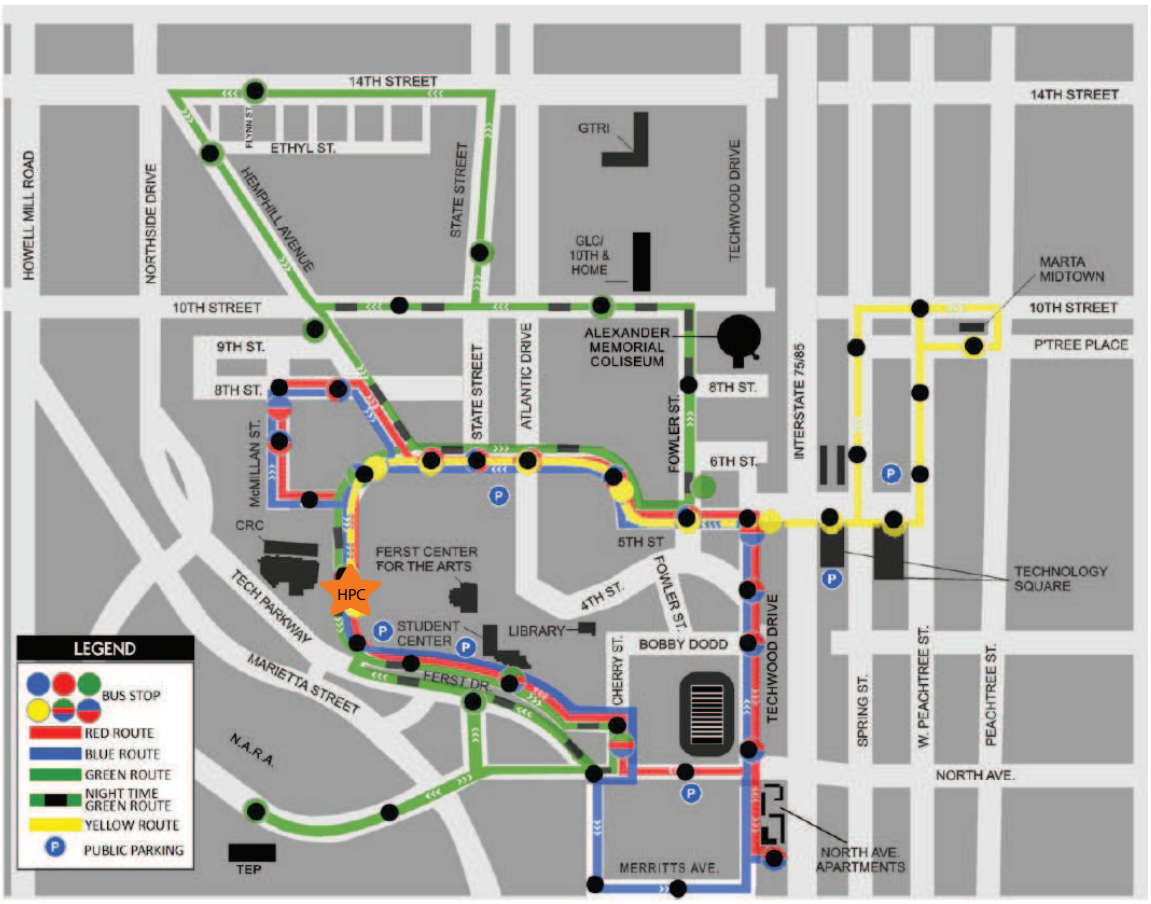


Figure 31: Georgia Tech bus route map. We have used the stations,

moment also means a low batching window. One can have more tasks arrive and hence imply a more conservative batching in the system.

In the experiments presented below and for the cases of non-controlled mobility, we assume that the vehicles follow the routes of Tech Trolley (Yellow), Green, and Blue buses in Figure 31. These vehicles follow their routes indefinitely and provide computation or communication services as they move[3].

Durations of the tasks are derived from the CERIT-SC workload log [3] which is a list of computational jobs submitted to the MetaCentrum distributed computing infrastructure[4] in the year 2013. Since some of these tasks were too short and many

[3]This assumption holds for the purpose of this simulation. In reality, the period of service for the vehicles must be much longer than the periods of the running of the algorithm for this assumption to hold

[4]`http://metavo.metacentrum.cz/en/index.html`

of them were too long (taking up weeks to complete on the MetaCentrum infrastructure), we filtered the dataset to keep only tasks with durations between 60 *sec* and 3600 *sec*. Tasks that take longer than this to complete processing are usually above the capability of a system with few processors, as in our cases. We have also kept task processing time above a minimum value to avoid skewing the system to mostly consist of very short tasks. For such cases, we expect the problems to be reduced to variations of TSP. Tasks are assumed to have no deadline unless stated otherwise.

4.4.2 Effect of Controlling Mobility versus MHPC/MF Speed

In this section, we run various experiments to understand the effects of the vehicle speed on the performance of the heuristics. The goal of this section is to understand how much advantage is gained when we control the mobility as the vehicles move faster.

The experiment setting is similar to the scenario described in Section 4.4.1. There are 3 vehicles serving the system described in Figure 31. We generate 600 batches of one or two tasks from a Poisson process. The approach taken for deriving the pickup and delivery location and duration of these tasks is described in Section 4.4.1. The inter-arrival time between these tasks is an exponential process. The mean inter-arrival time for the tests that use the MHPC framework is 250 *sec* and for the tests that use the HPC+MF framework is 150 *sec*. The reason for this difference is that in general, the MHPC framework is capable of handling systems of higher capacity due to the fact that tasks immediately arrive to the processing queue compared to the HPC+MF framework where tasks need extra travel on the MFs to arrive into the HPC's processing queue. We have chosen these values so that each system performs in an area that it is neither performing over nor under capacity. In each experiment, we vary the vehicle speed from 2 m/s to 40 m/s and observe the effects. The cases that test MHPC framework consider 2 processors on each MHPC. The cases that

consider the HPC+MF framework consider a single HPC with $3 \times 2 = 6$ processors placed on the designated location on Figure 31. We have run each of the tests 20 times, each time generating a new random set of data, and have averaged the results. Results are shown in Figures 32 and 33. Each of these figures compares two cases. These cases are described below:

- **Controlled Mobility:** We use the heuristics of Section 2.7 for the MHPC system and heuristics of Section 3.6 for the HPC+MF system. In both of these cases, we control the mobility of the vehicles to gain a better objective value, which is completing the service for each bath of arriving tasks and the previously awaiting tasks as soon as possible (refer to definition of RT in Section 2.2.3 and 3.2.3). These cases cover the first column of Table 3.

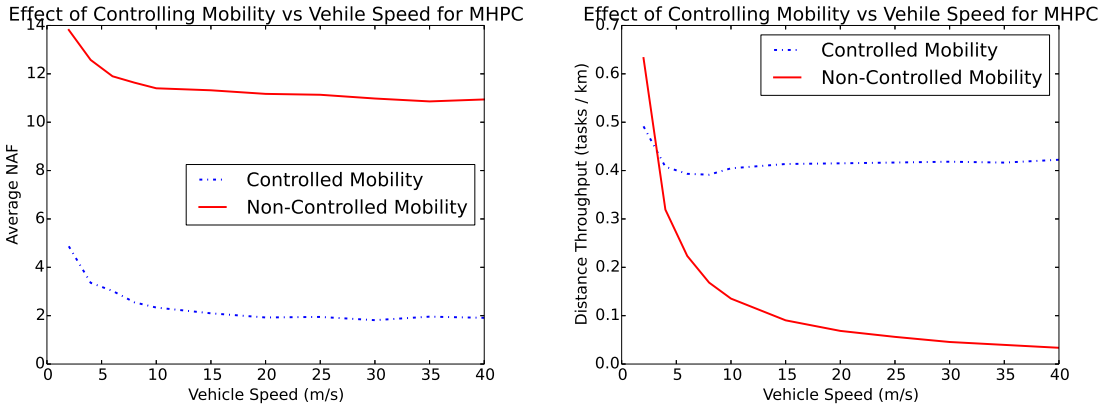
- **Non-controlled Mobility:** we use the heuristics of Section 4.2 for the MHPC system and heuristics of Section 4.3 for the HPC+MF system. As stated in Section 4.4.1, we assume that the three vehicles in these cases follow the blue, green, and yellow routes shown on Figure 31. These cases cover the second column of Table 3.

Figure 32 shows the comparison of the controlled mobility and non-controlled mobility case for the MHPC system (first row Table 3). We first compare the NAF value of the overall solution for the two cases in Figure 32a. It is evident from the figure that controlling the mobility significantly boosts the quality of the service that tasks receive, measured by NAF. This improvement is 6-fold in this case. It is also evident that in both cases of controlled and non-controlled mobility, increase of speed beyond some point does not improve the service. Finally, it is also notable that the difference between the controlled mobility and non-controlled mobility is fundamental and has its roots in the dynamics of the heuristics behind each in a way that even the highest speed for the non-controlled mobility does not perform as goos as the lowest

speed for the controlled mobility case. This means that switching to controlling mobility, if possible, is a much better recommendation for improving the performance of the system than increasing the speed.

Figure 32b compares the value of distance throughput (defined in Section 2.2.2) for the cases of controlled mobility and non-controlled mobility. Note that while a higher distance throughput means that the system is capable of serving the tasks while making less travels, it is not an objective of neither the heuristics of Section 2.7 for the controlled mobility nor the heuristics of Section 4.2 for the non-controlled mobility case. Figure 32b shows that while for the lowest speed of 2 m/s the HPC+MF provides a better service, the two cases quickly diverge with the controlled mobility case always outperforming the non-controlled mobility case thereafter. To understand the above observations, first note that for the very low speeds while distance that all MHPCs cover is less, their movements are not coordinated towards better serving the tasks. Instead in the controlled mobility case, the MHPCs slightly longer travels are coordinated to serve the tasks. This implies that even in the slower speeds, the MHPCs that provide service travel less, but the total trip of all MHPCs is shorter for the non-controlled mobility case. As the speeds grow, the total distance of the MHPCs in the non-controlled mobility grows much higher than the controlled mobility. This is due to the fact that in the non-controlled mobility, they cover more distances on their pre-determined route as their speed grows and in contras to the controlled mobility case, there is no correlation between the covered distance and the provided service.

Figure 33 shows the comparison of the controlled mobility and non-controlled mobility case for the HPC+MF system (second row Table 3). We first compare the NAF value of the overall solution for the two cases in Figure 33a. It is evident from the figure that controlling the mobility significantly boosts the quality of the service that tasks receive, measured by NAF in this case as well. This improvement is almost 15-fold in this case. Similarly, in both cases of controlled and non-controlled



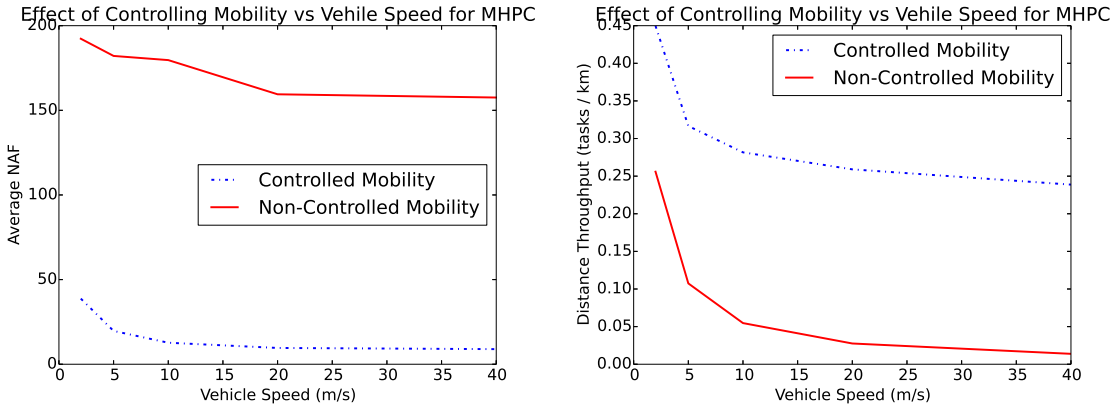
(a) Effect on average objective value. **(b)** Effect on distance throughput.

Figure 32: Effect of controlling speed versus vehicle speed on the objective value and distance throughout for the MHPC system.

mobility, increase of speed beyond some point, 10 m/s in this case, does not improve the service. Finally, we observe the same fundamental difference between the performance of the controlled and non-controlled mobility case. This means, similar to the MHPC instance, switching to controlling mobility, if possible, is a much better recommendation for improving the performance of the system than increasing the speed.

Figure 33b compares the value of distance throughput (defined in Section 3.2.2) for the cases of controlled mobility and non-controlled mobility. This figure shows that in contrast to the MHPC system of Figure 32b, in the HPC+MF system, the controlled mobility case always has the superiority in distance throughput as well. To understand the this observation, note that starting of any processing in the HPC+MF system requires the MFs to visit the HPC location. This incurs significant extra travel, specially on the non-controlled mobility case as the HPC location will be only visited once per pre-determined route. This makes the non-controlled mobility case to have extra rounds and deteriorates its distance throughput quickly. As the speeds grow, similar to the MHPC case, the total distance of the MFs in the non-controlled mobility grows much higher than the controlled mobility. This is due to the fact that in the non-controlled mobility, they cover more distances on their pre-determined

route as their speed grows and in contras to the controlled mobility case, there is no correlation between the covered distance and the provided service.



(a) Effect on average objective value. **(b)** Effect on distance throughput.

Figure 33: Effect of controlling speed versus vehicle speed on the objective value and distance throughout for the HPC+MF system.

4.4.3 Effect of Controlling Mobility versus Task Arrival Rate

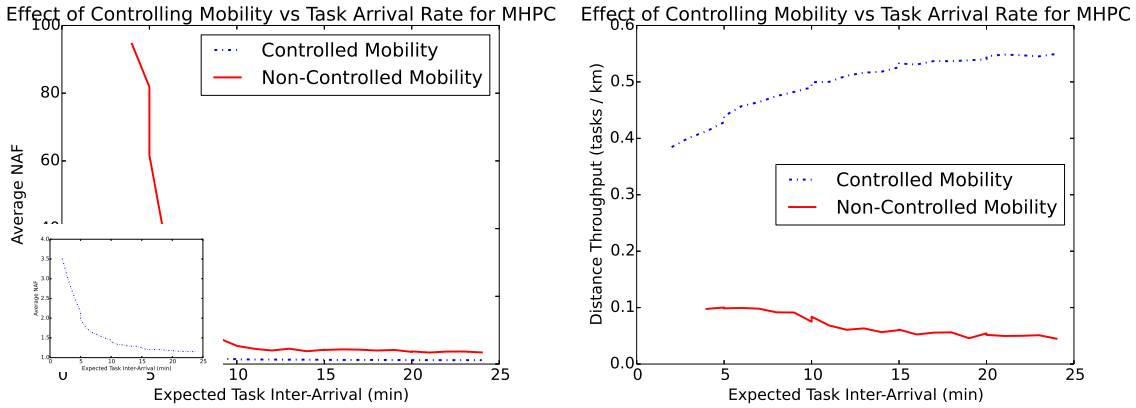
In this section, we run various experiments to understand the effects of the task arrival density on the performance of the heuristics. The goal of this section is to understand how much advantage is gained when we control the mobility as the tasks arrive denser or sparser from each other.

The experiment setting is similar to the scenario described in Section 4.4.1. There are 3 vehicles serving the system described in Figure 31. The cases that test MHPC framework consider 2 processors on each MHPC. The cases that consider the HPC+MF framework consider a single HPC with $3 \times 2 = 6$ processors placed on the designated location on Figure 31. We generate 600 batches of one or two tasks from a Poisson process. The approach taken for deriving the pickup and delivery location and duration of these tasks is described in Section 4.4.1. The inter-arrival time between these tasks is an exponential process. The mean inter-arrival time for the tests that use the MHPC framework is changed from 250 *sec* to 1500 *sec* for the MHPC tests and from 150 *sec* to 1500 *sec* for the HPC+MF tests in 50 *sec* increments. We have

run each of the tests 20 times, each time generating a new random set of data, and have averaged the results. Results are shown in Figures 34 and 35. Each of these figures compares two cases of controlled mobility and non-controlled mobility similar to those described in Section 4.4.2.

Figure 34 shows the comparison of the controlled mobility and non-controlled mobility case for the MHPC system (first row Table 3). We first compare the NAF value of the overall solution for the two cases in Figure 34a. It is evident from the figure that controlling the mobility significantly boosts the performance specially for densely arriving tasks. In fact, this difference is so huge that we needed to include the smaller figure inside Figure 34a to magnify the behavior of the NAF for the controlled mobility case. It is also observed that this difference in performance, while still present, is much less if tasks arrive more sparsely. The reason is that for sparsely arriving tasks both systems have much more time to react to the arriving information and the difference in planning (controlled mobility) vs piggybacking (non-controlled mobility) of computation becomes less evident.

Figure 34b compares the value of distance throughput (defined in Section 2.2.2) for the cases of controlled mobility and non-controlled mobility. It is observed that the distance throughput for the controlled mobility case always shows superiority to the non-controlled mobility case. Beyond this, we also observe that the distance throughput increase for the controlled mobility case as the tasks arrive more sparsely. This is due to the fact that with more sparse tasks the controlled mobility system can better plan to serve tasks that have lesser accumulation in the processors' queues and it also avoids any unnecessary travels. The non-controlled mobility case on the other hand does not appear to perform better even if the tasks are arriving sparser since it does not use the more planning time to the benefit of the system and always relies on the pre-determined routes of the MHPCs to serve no matter how dense or sparse the tasks arrive.

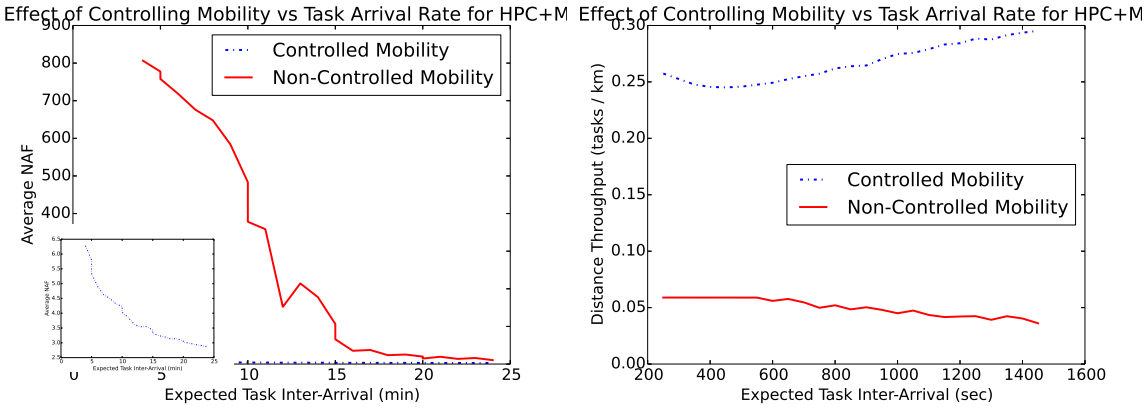


(a) Effect on average objective value. **(b)** Effect on distance throughput.

Figure 34: Effect of controlling speed versus task arrival rate on the objective value and distance throughout for the MHPC system.

Figure 35 shows the comparison of the controlled mobility and non-controlled mobility case for the HPC+MF system (second row Table 3). We first compare the NAF value of the overall solution for the two cases in Figure 35a. The trend is similar to those observed for the MHPC system except that ue to the requirement of visiting the HPC for processing any picked up task and since the non-controlled mobility system visits the HPC location once per round of visit of locations per route, the opportunity to start processing of the tasks occur with extra delays and hence the service given to the tasks is evn worse resulting in even more divergent performance compared to the controlled mobility system whose performance is magnified in the smaller figure inside Figure 35a.

Figure 35b compares the value of distance throughput (defined in Section 3.2.2) for the cases of controlled mobility and non-contsrolled mobility. The behavior is similar to that observed in Figure 34b with the increase in performance for the controlled mobility system and non-difference for the non-controlled mobility system. The reasoning behind the observed behavior is similar.



(a) Effect on average objective value. **(b)** Effect on distance throughput.

Figure 35: Effect of controlling speed versus task arrival rate on the objective value and distance throughout for the HPC+MF system.

4.4.4 Effect of Computation on the Move

In this section we compare the MHPC and HPC+MF heuristics. This is a comparison along the first row of Table 3. The MHPC framework allows computation on the move as the MHPCs can start processing tasks once they are picked up and as they are moving towards other tasks for pickup and delivery, they can process the tasks that they have already picked up. The HPC+MF framework, on the other hand does not enjoy the computation while moving paradigm and instead separates the communication and computation components. In this framework, picked up tasks shall be dropped off at the stationary HPC for processing and the results shall be received back from the HPC before delivery. While the HPC can process these tasks in the same time that the MFs are off to other pickup/deliveries, there is an explicit requirement to visit the HPC location in this framework. It is expected that this requirement results in an inferior performance compared to the MHPC framework.

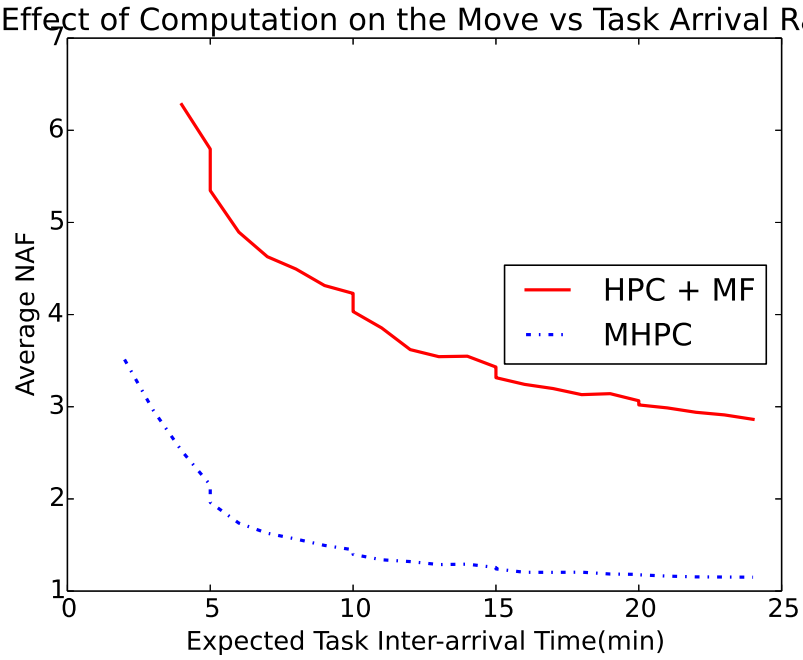
We start this section by comparing the performance of the two systems for similar batches of tasks that arrive with various rates. The goal of this first experiment to understand the fundamental performance differences between the two systems. The experiment settings are mostly the same as Section 4.4.3 for both the MHPC and

HPC+MF frameworks. The mean inter-arrival time for the tests that use the MHPC framework is changed from 250 *sec* to 1500 *sec* for the MHPC tests and from 150 *sec* to 1500 *sec* for the HPC+MF tests in 50 *sec* increments. To keep the MHPC and HPC+MF frameworks of this experiment comparable, we use 3 MHPCs each with 2 processors and compare it with a system of 3 MFs and an HPC of $3 \times 2 = 6$ processors. This way, the computational capacity of the two systems are comparable and we can focus on contrasting the fundamental difference of them.

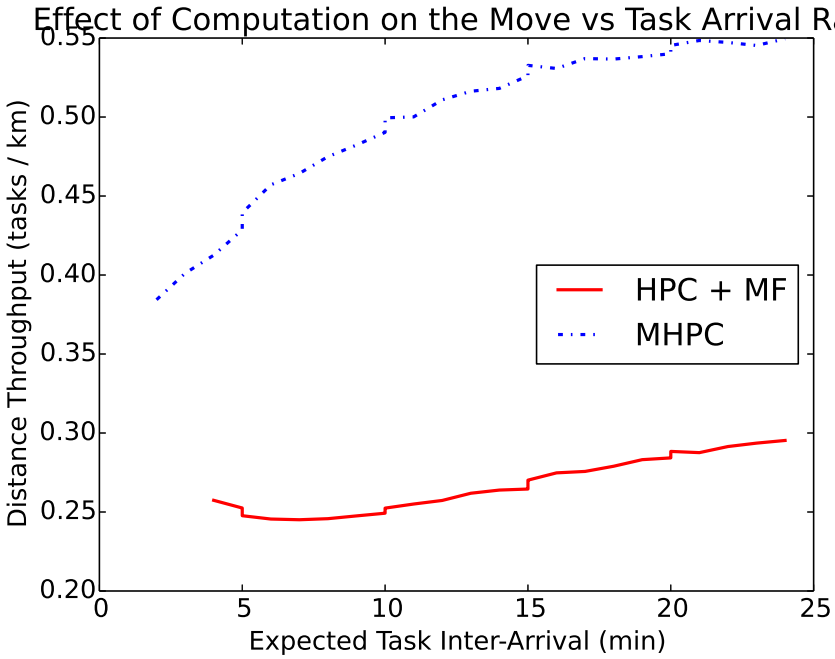
Figure 36a shows the NAF versus the arrival rate for both MHPC and HPC+MF systems. While both perform better with tasks arriving more apart from each other, it is obvious that the MHPC heuristic always outperforms the HPC+MF. This can be seen when one notices the fundamental difference of the two frameworks. Under similar scenarios, the computation on the move is always an advantage allowing to provide better service. One may also compare Figure 36a with Figures 34a and 35a. While both computation on the move and controlling mobility provide an edge in performance the latter provide a much larger advantage compared to the former. This means that in general changing a non-controlled mobility system to a controlled mobility system by using heuristics of Chapter 2 or 3 is a better investment than changing an HPC+MF to an MHPC system that allows computation on the move.

Figure 36b shows the distance throughput versus the arrival rate for both MHPC and HPC+MF systems. It is observed that both systems eventually present an increase in distance throughput as tasks arrive more sparsely which stems from the fact that both MHPC and HPC+MF frameworks apply control over mobility of the vehicles. It is also observed that the MHPC system always makes less trips to serve the same tasks since it does need the extra trip to the HPC location.

In the second experiment of this section, we make an attempt to understand the reason for the performance gap observed in Figure 36 and to understand if there



(a) Effect on average objective value.



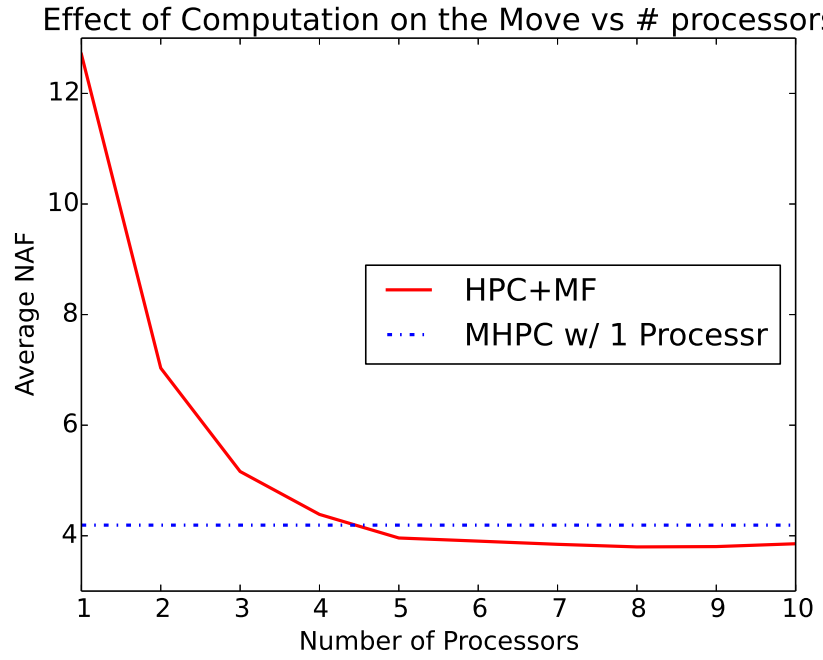
(b) Effect on distance throughput.

Figure 36: Effect of computation while moving versus task arrival rate on the objective value and distance throughput.

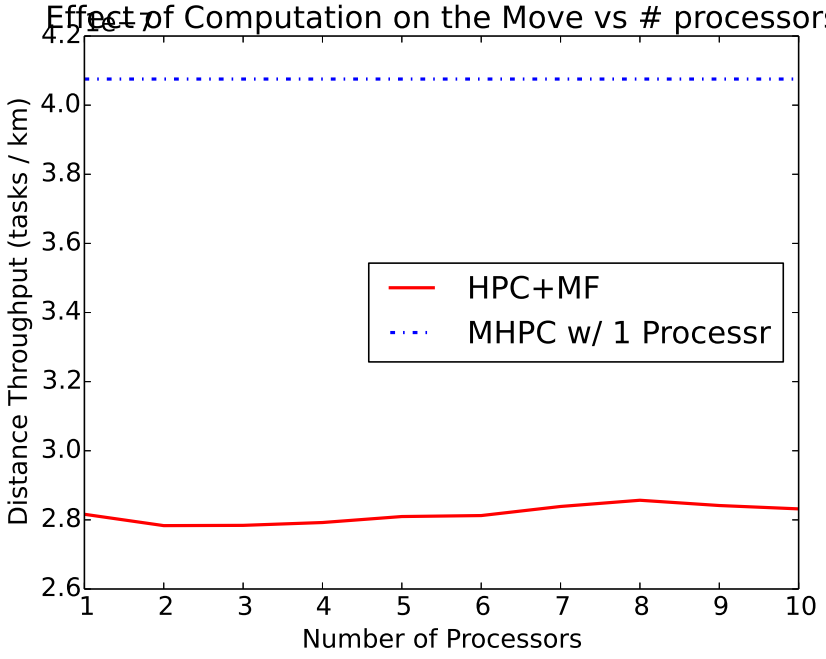
are alternative approaches to upgrade an HPC+MF system in order to provide comparable service to an MHPC system. In this experiment, we add processors to an

HPC+MF system and compare it with a baseline MHPC system that employs a single processor. Both frameworks use 3 vehicles (MFs or MHPCs) and they both serve 600 tasks generated from an Poisson process with a 200 *sec* inter-arrival time and similar settings to those described in Section 4.4.1. Figure 37 provides comparisons for objective value and distance throughput. It shall be noted that the MHPC system always has a single processor and the x-axis only shows the increase of processors for the HPC+MF and not for the MHPC system. The goal is to observe if increasing the processors of an HPC+MF system and hence its computation capabilities can remedy its deficiency of computation on the move when compared to a baseline MHPC system. Figure 37a shows that while initially the MHPC system performs better than the HPC+MF with a 3-fold advantage in the case of similar single-processor systems, eventually the HPC+MF catches up in performance. In specific, in Figure 37a and HPC+MF system of 5 processors can perform slightly better than the MHPC system. This shows that if mounting the computation resources on the vehicles in an infeasible proposal for boosting the performance of a system one can alternatively provide a higher performing stationary computation resource and get similar performance in some cases. Figure 37b shows that the MHPC system always serves more tasks with the same travel in this case. This is obvious as increasing the number of processors is not expected to affect the trips that the vehicles make. In fact, in Figure 37b the HPC+MF system does not exhibit any better performance with number of processors and hence if one is concerned about the HPC+MF system making more trips due to the requirement to visit the HPC, increasing the number of processors cannot help remedy for the more trips.

In the third experiment of this section, we make the MFs serving in an HPC+MF system increasingly faster and compare it with a baseline MHPC system that employs MHPCs moving at a very slow speed ($2\ m/s$) another baseline MHPC system



(a) Effect on average objective value.



(b) Effect on distance throughput.

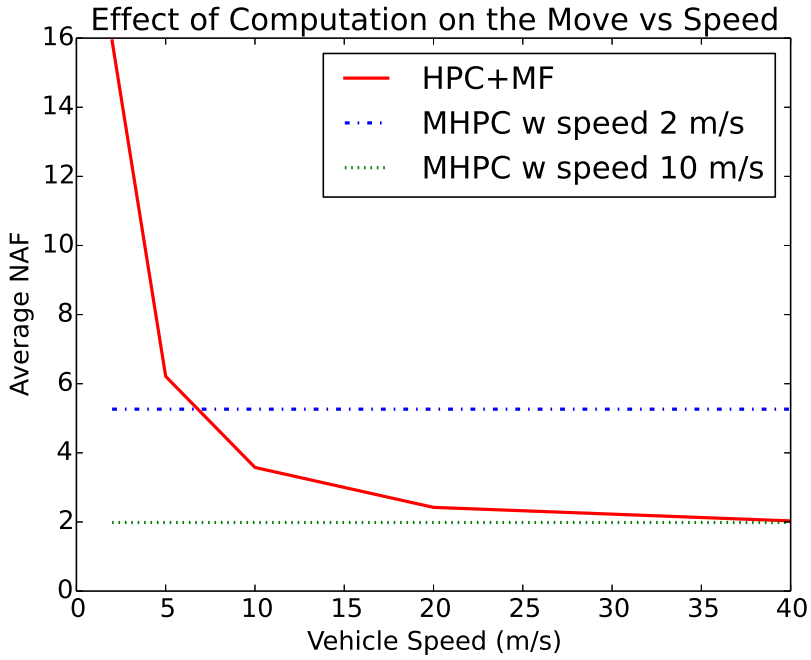
Figure 37: Effect of computation while moving versus number of processors on the objective value and distance throughput. The MHPC plot shows the baseline case for a system consisting of three MHPCs with a single processor.

that employs MHPCs moving at a regular speed (10 m/s). Similar to previous experiment, both frameworks use 3 vehicles (MFs or MHPCs) and they both serve 600 tasks generated from an Poisson process with a 200 sec inter-arrival time and similar settings to those described in Section 4.4.1. Figure 37 provides comparisons for objective value and distance throughput. It shall be noted that the MHPC system always moves at a speed of 2 m/s or 10 m/s in the two baseline cases and the x-axis only shows the increase of speed for the MFs for the HPC+MF and not for the MHPC system. The goal is to observe if increasing the speed of a MFs and hence the communication capabilities of the HPC+MF system system can remedy its deficiency of computation on the move when compared to a baseline MHPC system. Figure 38a shows that while initially the slow-moving MHPC system (2 m/s) performs better than the HPC+MF with a 3^{+}-fold advantage in the case of similar HPC+MF system, eventually the HPC+MF catches up in performance as the MFs move faster. In specific, in Figure 38a and HPC+MF system with MFs moving at 10 m/s can perform more than twice better than the MHPC system. As for the regular-moving MHPC system (10 m/s) the HPC+MF system can only catch up at very high speeds of 40 m/s. This shows that if mounting the computation resources on the vehicles in an infeasible proposal for boosting the performance of a system, another alternative is to provide faster moving vehicles. Figure 38b shows that the MHPC system makes less trips than the HPC+MF in general. The only exception is for very low speed of 2 m/s on the HPC+MF system having better performance than both MHPC systems. We account this to the fact that in this case the HPC+MF makes very conservative visits to the HPC location, which can also be a task location as well in general and in both the HPC+MF and MHPC frameworks, and since the cost of too many visits to the HPC location is higher for the HPC+MF location in terms of service quality (NAF) there is less trips made in the HPC+MF system. Despite this, soon after increasing the MF speed, the HPC+MF makes more aggressive visits to the HPC location as it

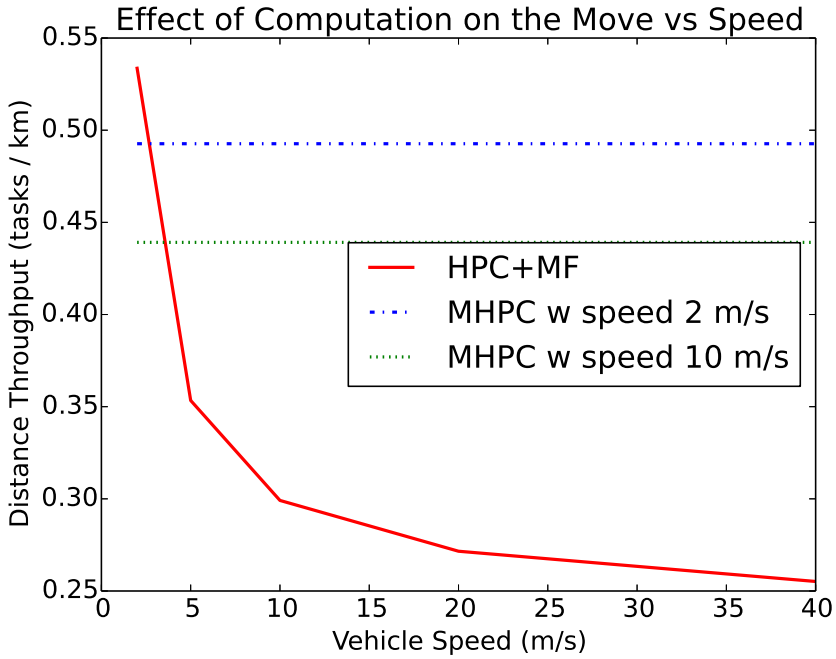
incurs less cost on the service and hence its total trips grows higher than the baseline MHPC systems. It also shows that although with increase of speed both MHPC and HPC+MF make more aggressive trips, but the HPC+MF utilizes the ability to have faster moving MFs more aggressively as visits to the HPC location on high speeds can significantly help the performance of the system.

4.5 Conclusions

Motivated by the Node-Initiated Message Ferrying (NIMF) proposed by Zhao et al., [57], we have explored the dimension of non-controlled mobility on the heuristics of Chapter 2 and 3. We have further categorized the algorithms on the dimension of computation on the move which is the distinction between the MHPC and HPC+MF problems. After elaborating on how these dimensions differ, we have developed heuristics that utilize pre-determined mobility of vehicles to piggyback a computation service (Section 4.2) or a communication service (Section 4.3) on them. We have compared these heuristics to their counterparts in Sections 2.7 and 3.6, respectively. Through the experiments, we have demonstrated the value in controlling mobility and that it brings significant performance gains to the system. This establishes the value of our work in Chapter 2 and 3 compared to the related work that does not attempt to control the mobility and instead relies on the pre-determined movement of network elements. Finally, we have demonstrated that there is also value in using the MHPCs compared to the combination of HPC and MFs. This is due to the fact that the MHPC combines the computation and communication services while the HPC+MF separates the two between the HPC and the MFs. We have shown that in this case, it is more feasible to make up for the lack of computation on the move by adding either more processors to the HPC and hence making computation more powerful or by having faster moving MFs and hence improving the communication capabilities.



(a) Effect on average objective value.



(b) Effect on distance throughput.

Figure 38: Effect of computation while moving versus vehicle speed on the objective value and distance throughput. The MHPC plot shows the baseline case for a system consisting of three MHPCs moving at 2 m/s and another with three MHPCs moving at 10 m/s.

Algorithm 7: Heuristic for HPC+MF problem with non-cotrolled mobility

Input: Groups of tasks: (t_i,G_{t_i}), Number of MFs: V, Number of HPC Processors: m, MF Routes: $\{L_i \; \forall i \in MF \; route\}$

Output: *Solution* = MF Assignment (per instance): $\pm i_k \; \forall k \in V \; , \forall i \in tasks$, Task Execution Schedule (per instance): $sched_k = \{((t_1, t_2), i)\} \; \forall k \in V$

```
1  Initialization:
2  for Each MF k ∈ V do
3  |  Pickup List = {};
4  |  Delivery List = {}.
5  end
6  for Each MF k ∈ V do
7  |  while True do
8  |  |  Visit the next location, L_k, on MF route.
9  |  |  if L_k is not the HPC location then
10 |  |  |  for Each task i in the MF's Pickup List do
11 |  |  |  |  if Pickup location of i, LP_i == L_k then
12 |  |  |  |  |  Add the task to Pickup List ;
13 |  |  |  |  end
14 |  |  |  end
15 |  |  |  for Each task j in the MF's Delivery List do
16 |  |  |  |  if Delivery location of j, LD_i == L_k then
17 |  |  |  |  |  if Current Time ≤ Deadline for Task j, D_i then
18 |  |  |  |  |  |  Mark task j as served.
19 |  |  |  |  |  else
20 |  |  |  |  |  |  Mark task j as rejected.
21 |  |  |  |  |  end
22 |  |  |  |  end
23 |  |  |  end
24 |  |  |  else if L_k is not the HPC location then
25 |  |  |  |  for Each task i in the MF's Pickup List do
26 |  |  |  |  |  Add i to the queue for HPC's FCFS scheduler.
27 |  |  |  |  end
28 |  |  |  |  Pickup List = {} for Each task j in the in HPC's ready queue
   |  |  |  |  (finished tasks) do
29 |  |  |  |  |  if MF k visits delivery location LD_j or task j then
30 |  |  |  |  |  |  Add j to Delivery List of MF k.
31 |  |  |  |  |  end
32 |  |  |  |  end
33 |  |  |  end
34 |  |  |  if all tasks are served an no further tasks are supposed to arrive to the
   |  |  |  system then
35 |  |  |  |  Terminate
36 |  |  |  end
37 |  |  end
38 end
39 return Solution= {MF Assignment, FCFS Schedule for HPC}, Rejected tasks.
```

CHAPTER V

PLAUSIBLE MOBILITY INFERENCE FROM WIRELESS CONTACTS USING OPTIMIZATION

5.1 Introduction

In this chapter, we introduce utilities that help further the research on computational ferries and message ferry-assisted computational clouds. One of the first assumptions in the framework of problems described in Chapter 2 and 3 that can be challenged is the fact that either user nodes are stationary or the locations where tasks/results are exchanged is a fixed location. In a more general case, user nodes can be truly mobile and the MHPCs or MFs shall account for this mobility when they attempt to exchange task/results with them. This mobility can be handled with any of the three following assumptions:

- One can assume that location and mobility of all user nodes are known at any point of time in the future. This assumption is the least realistic handling of mobility and can be reasonable only if user nodes have a predictable mobility pattern..
- One can assume that MHPC or the MF can always inquire about the current location of the the user nodes and then track them with persistent update requests until they can meet or get within range of the user node for information exchange.
- One can assume that the user nodes either broadcast their location periodically or assuming that they move with some average speed, they only broadcast any change in their direction. These broadcasts can be received by MHPCs or MFs

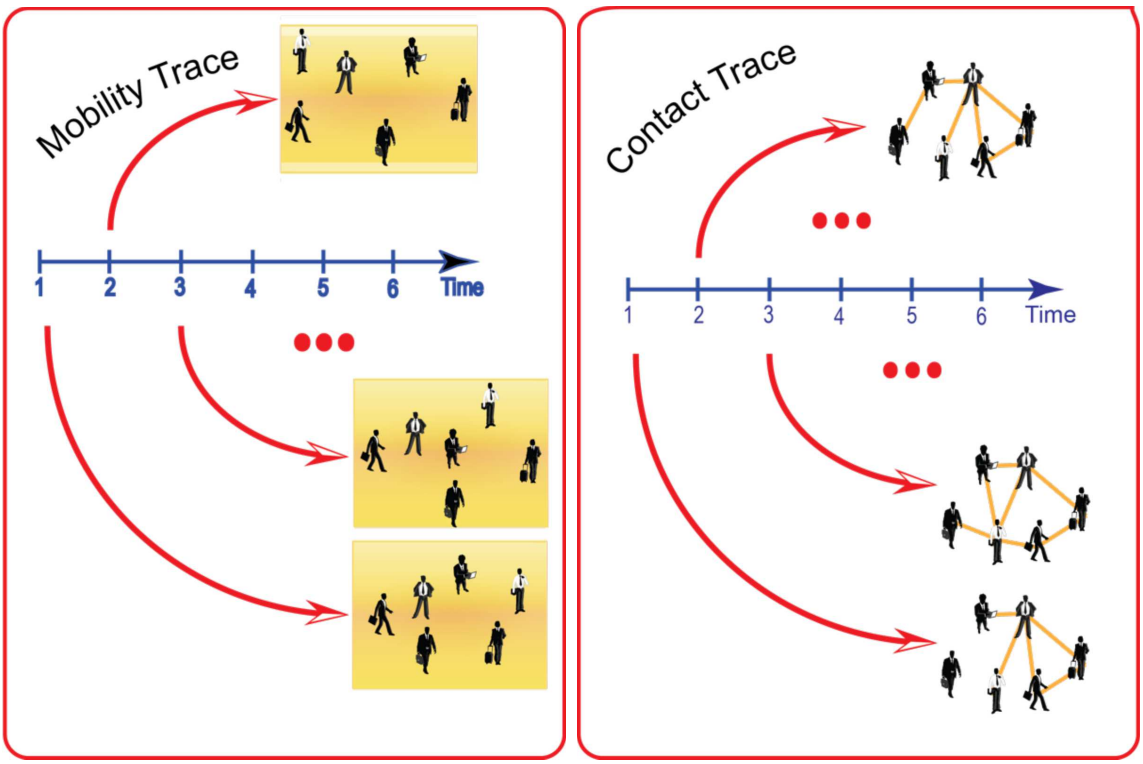
and they can adjust their mobility to meet with the user node accordingly.

All of these assumption require access to some sort of mobility trace. In this chapter, we argue alternate ways to collect and maintain mobility traces. Contributions of this chapter can be used as a tool that helps better integrate mobility into these frameworks.

In general, studies of mobile wireless protocols benefit from real-world traces that measure and record node locations over time. These *mobility traces* can be combined with radio models to produce network connectivity for simulation studies. Measuring node locations with fine-grained time and space granularity is challenging and hence the number of real world datasets on mobility that are publicly available is not significant. Examples of mobility traces are reported by Rhee et al. [39] with scenarios in a university campus and in public places and by Banerejee et al. [8] for public transit buses in a campus. A typical mobility trace that records snapshots of nodes' mobility over time is depicted in Figure 39b.

As an alternative, *contact traces* measure and record pairwise node connectivity over time and do not require accurate positioning systems. Since they come from radio transmissions, contact traces are able to capture environmental effects on communication capability such as obstacles. These effects are mostly missing in mobility traces. Contact traces record on-off timing information about links, hence they can be more compact than mobility traces that record location data at every time instant. A typical contact trace that records the pairwise proximity-based connections among nodes in depicted in Figure 39a.

Contacts are typically recorded by using Bluetooth or WiFi in an infrastructureless mode such that all devices in range are seen as a contact. In such experiments, mobile devices equipped with a wireless interface are attached to moving entities such as people or vehicles. These mobile devices sample their environments periodically and log the presence of neighboring nodes. At the conclusion of the experiment these



(a) A mobility trace records node mobility over time.

(b) A contact trace records pairwise node connectivity over time.

Figure 39: An illustrative description of mobility traces and contact traces.

logs are collected and a contact trace is constructed from them. Contact trace collection and analysis is reported in the literature and is used extensively to study mobile networks for various settings ranging from a campus scenario [33][49] to urban scenarios [36] or a disaster area [5]. Aschenbruck et al. [6] summarize these datasets. While contact trace collection is easier than mobility trace collection, it is still considered a challenging task. It requires instrumenting nodes with collection capability and conducting a well-planned and coordinated collection "run".

Despite collection difficulties, contact traces remain more available to researchers than mobility traces. However, mobility traces offer an opportunity for more detailed simulations and can be used to explore new scenarios, for example by combining mobility with alternative radio models to generate new contact traces. For these reasons, we are interested in the problem of inferring a *plausible mobility trace* from a

contact trace. A plausible mobility trace is one that can produce the given contract trace. In general, there are many mobility traces that are consistent with a given contact trace, and it is impossible to reconstruct exact locations. However, we believe that exact locations are not required in many experiments that deal with studying mobility. Our research explores the effect of this loss of accuracy in relative locations that results from our inference algorithm.

Few solutions are proposed for this specific problem. Wang et al.[51] and Ristanovic et al. [40] describe the inference problem and suggest some approaches. These works are evidence that the problem is of interest to the community but neither explores the solution in detail.

Whitbeck et al. [54] propose a complete algorithm for the mobility inversion problem. This algorithm is inspired by the work in dynamic graph drawing and is based on the concepts of physical forces. The goal of the algorithm is to satisfy three fundamental constraints on the mobility of the nodes. These constraints limit the speed of nodes and their distances as they setup and lose contact. To do this, the authors define three forces on each node based on their contact history and future; it is the equilibrium of these forces that gives a relative set of locations satisfying the given contacts.

A major disadvantage of the force-based algorithm is its dependence on many parameters that need to be tuned properly in order for the algorithm to provide accurate contact to mobility transformation. These include the rigidity constant, the damping factor, the intensity constant, cutoff distance, etc. that are used in the calculation of the forces. There is no general rule for tuning these parameters for the algorithm. Our work is motivated by the need to find a contact to mobility transformation algorithm that will work without the need for such parameter tuning.

Our proposed algorithm uses an approach based on formulating plausible mobility inference as a feasibility problem for a small set of fundamental constraints on mobility

of the nodes. Our algorithm provides more accurate results and is more robust against choices of parameters.

5.2 Mobility Inversion Algorithms

Let N be the number of nodes that reside in a $d = 2$ dimensional $L \times L$ square field that consists of T time instances. A contact trace in this setting corresponds to a $N \times N \times T$ matrix. This representation implies that we have access to snapshots of the connectivity structure of nodes at time instants $\mathcal{T} = \{t_o, ..., t_{T-1}\}$ where $|\mathcal{T}| = T$, i.e. we know the connectivity structure for T discrete snapshots[1]. A mobility trace is a $d \times N \times T$ matrix, X, where each element $x_i^k = (:, i, k)$ is the location of node i at time t_k.[2]. A summary of these parameters is provided in Table 4.

Table 4: Parameters used in the Mobility Inference Algorithm 8.

Parameter	Interpretation
N	Number of nodes
R	Transmission range.
v	Maximum speed
T	Number of snapshots of the trace
d	Number of dimensions
L	Size of the simulation field
t_k	Time instant of the k^{th} snapshot of the trace
x_i^k	Location of node i at time t_k

Our algorithm does not use any information about initial locations nor does it makes any assumptions about the underlying mobility behavior of the nodes. The only assumptions made are the following:

[1]Note that most real traces are in the form of a "continuous" contact trace in which start and stop times of contact events are recorded with some precision. Such contact traces can be transformed into our desired discrete notation by capturing connectivity structure of nodes at discrete time instants with a uniform step of $\Delta t = t_k - t_{k-1}$. The value of Δt needs to be chosen carefully to keep the contact trace reasonably small in size while capturing all contact events that are long enough.

[2]Throughout this paper, we use the notation of MATLAB-like colons,$(..., :, ...)$, to represent specific dimensions in a multidimensional matrix. For example $A(:, 2)$ corresponds to the second column in the matrix A.

- At any moment $t_k \in \mathcal{T}$ in time, two nodes i and j are in contact once their distance, $||x_i^k - x_j^k||_2$ is less than a given radio range minus a margin, $R(1-\epsilon)$, and are out of contact if this distance is larger than $R(1+\epsilon)$ (modified circular transmission range radio model).

- Nodes move slower than a given maximum speed v. This means that the distance between the location of a node i at time t_k and t_{k-1} ($t_k \in \mathcal{T} \setminus \{t_0\}$), which is represented as $||x_i^k - x_i^{k-1}||_2$ is smaller than the maximum possible distance that can be traveled, namely $v \times \Delta t$, where Δt is the time difference between snapshots.

- Nodes are always confined inside the $L \times L$ mobility field.

Now we describe the inference process followed by our algorithm. We use the notation $G = (V, E)$ for the evolving graph corresponding to the input contact trace matrix, C. In this notation $V = \{1, 2, \ldots, N\}$ is the set of all nodes. This evolving graph is a time-series of ordinary graphs $G^k = (V, E^k)$. An edge $e_{ij}^k \in E^k$ represents a connection between nodes i and j at time instant t_k. We associate graph G, with its adjacency matrix C, i.e., each of the graphs G^k has an adjacency matrix $C^k = C(:, :, k)$.

A high-level description of our algorithm is illustrated in Figure 40. The algorithm starts by finding an initial set of locations for nodes that is consistent with the initial state of the contact trace at time t_0, namely G^0 with adjacency matrix $C(:, :, 0)$. After this initial step, to find the locations of the nodes at time instants $t_1, \ldots, t_{T-1}$, we solve a feasibility problem[3] to find a set of locations for each of the snapshots in the contact trace corresponding to these time instants. This feasibility problem ensures

[3]We have tried various objectives to solve the problem as a complete optimization problem. Examples include minimizing nodes movement and minimizing node deviation from the center of the field. Despite this, our simulations show that the best results are achieved when the problem is stated as a feasibility problems. Besides this, there is no justification for an objective to be minimized in this problem.

consistency with the three fundamental assumptions mentioned earlier. We use the location of nodes at the previous step as the initial point for the next step, and solve the problem iteratively to find the location of the nodes at each of the snapshots. This process is described in Algorithm 8.

Algorithm 8: Mobility inversion algorithm.

Input: Contact Trace $C_{N\times N\times T}$, Transmission range R, Maximum speed v
Output: Mobility trace X_{final}

Initialization: Find a set of initial locations X_0;
$X_{final}(:,:,0) \leftarrow X_0$.
Optimization:
for $k := 1$ ***to*** $T-1$ **do**
  $X_{init} \leftarrow X_{final}(:,:,k-1)$;
  Solve the nonlinear optimization problem in Figure 40 with $C^k = C(:,:,k), X_{init}, R, v$ to find X;
  $X_{final}(:,:,k) \leftarrow X$.
end
return X_{final}

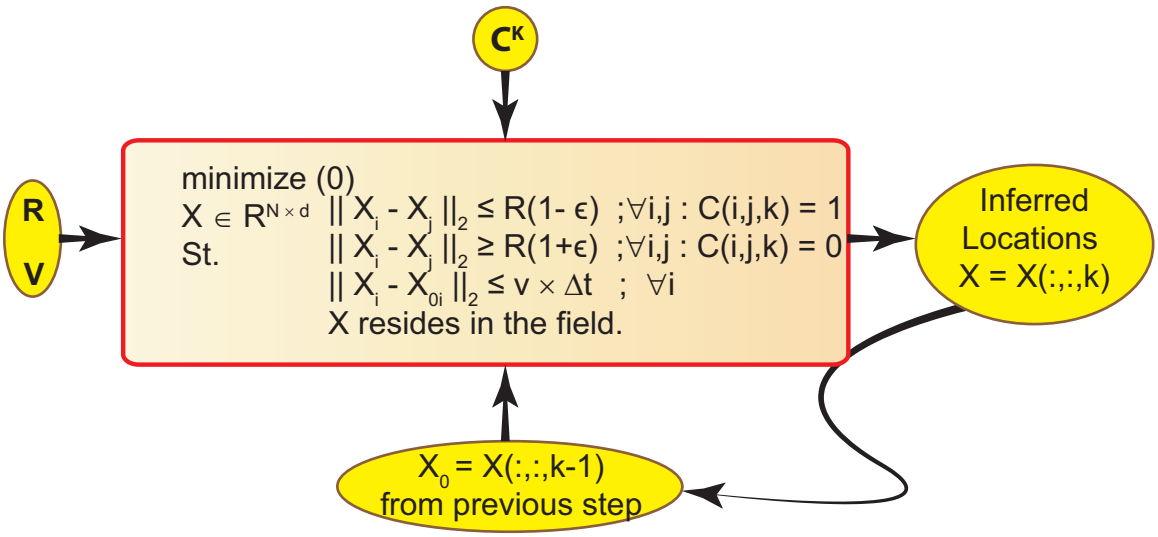


Figure 40: Optimization-based algorithm takes an initial point ($X_{init} = X(:,:,k-1)$), a contact trace ($C^k = C(:,:,k)$) along with its corresponding radio range (R) and a value of maximum speed (v) to infer a mobility ($X = X(:,:,k)$) in an iterative manner. Inferred locations for each time step are given as an initial point to the solver for the next step. This process is repeated for $t = t_0, \ldots, t_{T-1}$.

Choice of the Initial Value: To address the issue of choosing a suitable value for X_0, we use the same method as above, except that the maximum speed constraint

is ignored. Since the solver still needs an initial point to seek the stationary point of the subproblem solved at the first iteration of the algorithm, we construct a distance matrix by finding the shortest path among all node pairs in terms of the number of hops using the initial connectivity structure, G^0. Next, we multiply this hop-count matrix by the transmission range R, i.e., we assume two nodes m hops from each other are at a distance $m \times R$, and replace unreachable node pairs with a reasonably large distance. Finally, we apply the classical Multidimensional Scaling (MDS) [14] on this matrix to yield a decent initial point.

5.3 Evaluation

5.3.1 Evaluation Methodology

Similar to the work in [54] our evaluation is based on using the *"original" contact traces* derived from known *original mobility traces*. Our algorithm is applied to produce an *inferred mobility trace* from this contact trace. This in turn is used to derive an *inferred contact trace* using the known radio range. We then compare the original contact trace with this inferred contact trace to judge the accuracy of our algorithm. We use three levels of comparison. This process is illustrated in figure 41.

Mobility-level Comparison: Assuming that we have access to the mobility trace from which the input contact trace originated. We can compare the original mobility trace and the inferred mobility trace for each location. As a direct comparison of absolute locations will not bear much information, we compare the distance between each node-pair at each time step for both traces in a two-dimensional histogram.

Contact-level Comparison: This comparison evaluates how accurately our algorithm replicates the connectivity structure of the original contact trace in the inferred contact trace. We compare the evolving graphs corresponding to these two contact traces over time based on the number of *missing links*, i.e., links present in

the input contact trace and missing in the inferred contact trace, *extra links*, i.e., links only seen in the inferred contact trace, and the sum of these two numbers.

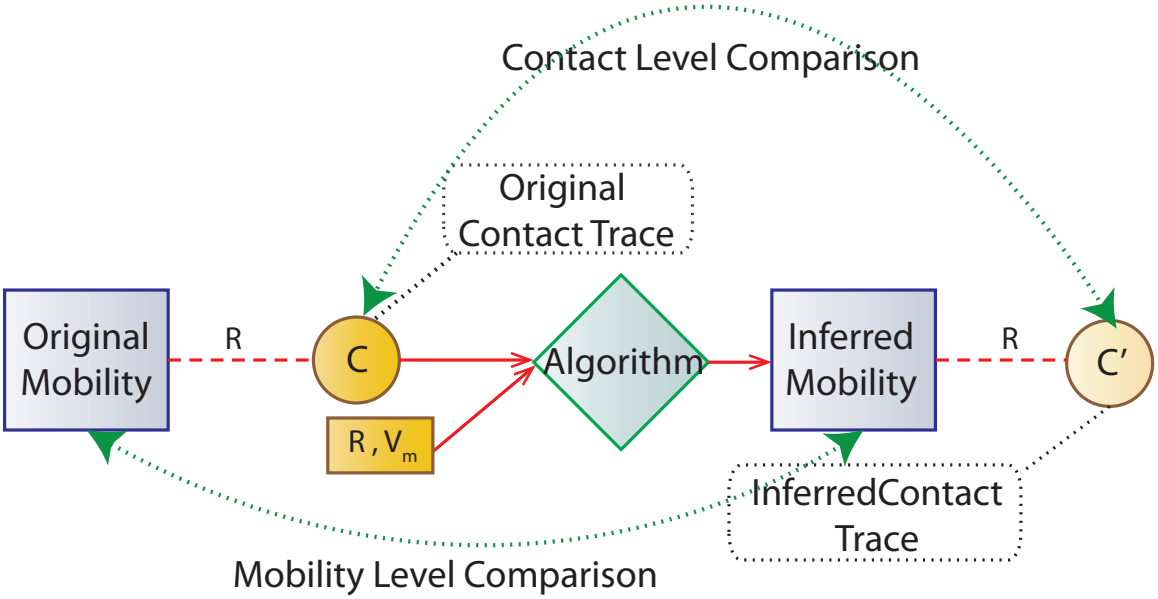


Figure 41: An illustration of concepts of original trace, inferred trace, original contact trace, and inferred contact trace. This figure explains which two traces are compared in mobility level and contact level comparisons.

Packet Delivery Ratio: This comparison involves running a simulation on the original and the inferred traces using the Opportunistic Network Environment simulator (ONE) [27]. In these experiments, we send random packets with a specific time to live (TTL) from a randomly chosen node destined for another randomly chosen node at every time step. Then, we compare the cumulative number of messages that reach their destinations before the TTL to total number of messages sent (packet delivery ratio) at every time step.

5.3.2 Evaluation Setup

We use three traces, two synthetic and one real, to evaluate our algorithm.

- *Random Waypoint (RWP)*[11], which is often used for simulation purposes in mobility literature despite its known limitations[55]. In this model speed, direction, and destination of each node are chosen randomly and independently

of other nodes. Our RWP instance consists of 50 nodes with a duration of 1000 seconds with snapshots taken every second, spanning over a $1000 \times 1000\, m^2$ field. Maximum speed is $10m/s$ and transmission range is set to $100m$.

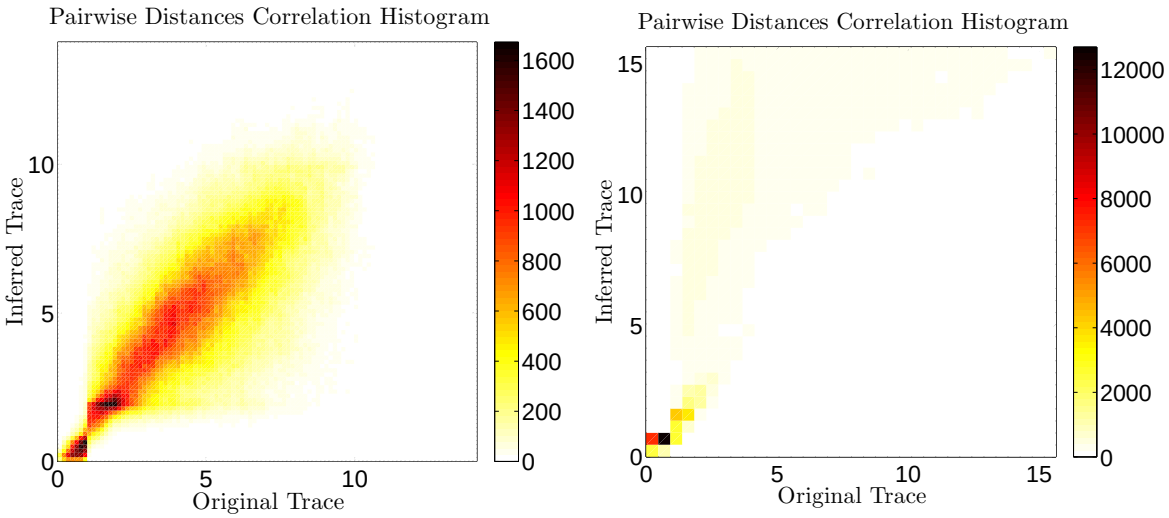
- *Self-similar Least Action Walk (SLAW)*[28], which tries to capture statistical patterns of human mobility including truncated power-law distribution of flights, pause-times and inter-contact times, and heterogeneously defined areas of individual mobility. One of the main features of this model is that it captures social contexts. The model is heavily based on GPS traces of human walks, including 226 daily traces collected from 101 volunteers in five different outdoor sites. Our instance of SLAW consists of 50 nodes simulated over 10 hours with snapshots taken every minute. The simulation field is $2000 \times 2000 m^2$; maximum speed is $10m/s$ and transmission range is $60m$. Other important parameters of the SLAW are 0.75 for the self-similarity of waypoints (on a scale of 0 to 1), and a minimum (maximum) pause of 30 seconds (1 hour) for the individuals chosen to match human mobility in a social setting.

5.3.3 Mobility-level Comparison

Figure 42 shows the histogram for pairwise distances normalized to the value of transmission range. We do not expect to see a perfect correlation between the original and inferred distances since the algorithm does not have any auxiliary information about the locations of the nodes. Instead, we observe a trend of large distances versus large distances and small distances versus small distances in the original and inferred traces respectively. Note that in the ideal case the histogram should have high values along its diagonal.

As the figures suggest, for RWP (Figure 42a), most of the distances are mapped accordingly in the inferred trace with a distribution of errors around the diagonal. For SLAW (Figure 42b), most of distances are small and mapped correctly (notice

the dark areas in the bottom left)and for a few larger distances(the lighter areas in the upper half of the figure) performance deteriorates.



(a) RWP **(b)** SLAW

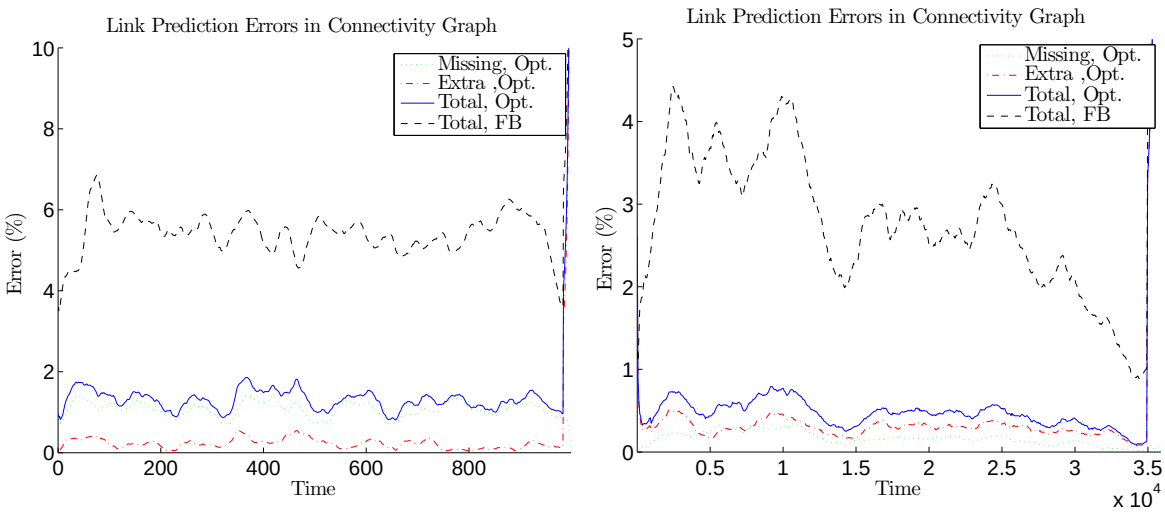
Figure 42: Pairwise distance histograms. Horizontal (Vertical) axis shows distance among node-pairs in the original (inferred) trace normalized by the value of the transmission range. Color intensities are proportional to the number of node-pairs having distances in the corresponding bin.

5.3.4 Contact-level Comparison

Figure 43 shows the contact-level errors. For a more meaningful comparison, we use the force-based heuristic of Whitbeck et al., [54] to infer the same trace. Except for the obvious choices like transmission range and maximum speed, we use the default values in the force-based heuristic. To preserve the readability of the figure, we plot the missing, extra, and total link errors for our optimization-based heuristic and only the total link errors for the force-based heuristic. For RWP, the figure suggests that most of the error is in the missing links which are present in the original contact trace but absent in the inferred contact trace. The total link errors are less than 2%, which corresponds to $0.02 \times 1225 \simeq 24$ errors out of the 1225 possible links.

For SLAW, although the trace is based on a much more complicated model and is hours long in duration with update intervals of minutes, Figure 43b suggests errors

lower than 1%, i.e., at most 12 links in error, almost all of which are missed connections as seen in the figure. This is 3 times lower than the error of the force-based heuristic.



(a) RWP **(b)** SLAW

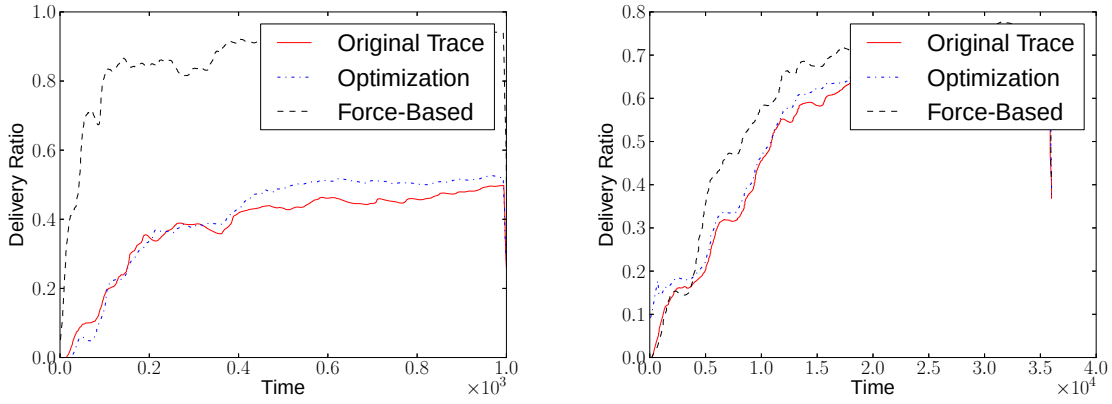
Figure 43: Contact-level errors over time. Extra link errors (Extra) are missing in the original, but present in the inferred contact trace. Missing link errors (Missing) are present in the original, but missing in the inferred contact trace. Total link errors (Total) is the sum of these. The horizontal axis represents time and the vertical axis shows the errors as percentage of the total number of possible links. The figure includes all three types of errors for the optimization-based algorithm (Opt.) and the total link errors for the force-based heuristic(FB).

5.3.5 Packet Delivery Ratio Comparison

Figure 44 shows the packet delivery ratio, calculated as the total number of packets received by a destination node divided by the total number of packets sent to that node up until the current time. In these experiments, we send a $1KB$ packet from a randomly chosen node to another randomly chosen node every second. The value of the TTL for messages is set to one-fifth of the trace length, e.g., 3 minutes for RWP. We use the Probabilistic ROuting Protocol using History of Encounters and Transitivity (PROPHET) [31] as the routing protocol for the simulation over simpler routing protocols like flooding in order to have a more realistic situation. Despite this, our complete set of results show that the choice of routing protocol has the

same effect on original and inferred contact traces and is irrelevant. We include the simulation results for the original trace, inferred trace using the optimization-based algorithm, and inferred trace using the force-based heuristic.

All figures suggest that the original and inferred traces show very similar behavior in packet delivery. The figures also suggest that the errors in Figure 43 do not significantly affect the packet delivery behavior. In all cases, force-based heuristic shows a higher packet delivery ratio, which means it has unnecessary links in the inferred trace (extra link errors). Also, the inferred trace using optimization-based algorithm always performs more closely to the original trace than the force-based heuristic with the difference most obvious in Figure 44a.



(a) RWP **(b)** SLAW

Figure 44: Message Delivery Ratio versus time. Vertical axis shows the total number of messages delivered until the current time divided by the total number of messages sent.

5.4 Conclusions and Future Work

In this chapter, we have explored an optimization-based algorithm to solve the problem of inferring mobility from contacts. Our algorithm accepts a contact trace that contains information about connectivity of nodes over time. Without any extra information or assumptions about the locations of these nodes, we infer a set of locations that could generate these contacts. Through extensive experiments with synthetic and real traces, we show that this inferred mobility has connectivity characteristics

that are comparable to the original trace and can be used in simulations instead of the original contact trace.

As stated earlier, our algorithm, unlike the work in [54] has the important advantage that it works without needing cumbersome and error-prone parameter tuning. The optimization framework on which our algorithm is built also has additional promise for future extensions including:

- In this work we have assumed a simple circular radio range model, i.e., two nodes are in contact whenever their distance is less than a given transmission range and disconnected otherwise. In reality, a contact trace should be interpreted along with its corresponding radio model which specifies the circumstances under which two nodes are actually in contact and factors such as the existence of obstacles, fading and shadowing, and inequality of signal strength inside the transmission range need to be taken into account. We believe our approach provides a suitable framework for incorporating a radio model since the feasibility constraints explicitly incorporate the radio range. One idea would be to replace R in the constraints with a random number drawn from a distribution of radio ranges derived from a specific radio propagation model.

- An input contact trace induces a set of constraints in our algorithm. Because of this, our algorithm can take as input multiple contact traces which simply translates into multiple sets of constraints for which an approximate feasible solution can be obtained. We believe this will allow us to produce more accurate contact to mobility transformation. If this is confirmed it may indicate that trace collection exercises can enhance their usability if they are augmented to obtain multiple traces simultaneously.

CHAPTER VI

CONCLUSION AND EXTENSIONS

In this thesis, we have explored the problem of using HPCs as means of computational offloading to help users dispersed in a geographical region with the intensive tasks on their hand-held devices. After motivating the problem of mobile offloading and reviewing its requirements, we have surveyed related literature in Chapter 1 of the thesis to understand the position of our research relative to similar problems. Specifically, we have laid the foundation of the problems considered in this thesis around scenarios where connectivity infrastructure is not available and size of the geographic region along with bandwidth considerations do not allow direct communication for all users. We have proposed and provided extensive study of the following two problems in Chapters 2 and 3:

- **Computational Ferrying via MHPCs (MHPC Problem):** This work has considered an algorithm for scheduling computation on a Mobile High Performance Computer (MHPC). We have suggested a framework where a number of User Nodes that have computationally expensive tasks will receive computation help from the MHPCs. These MHPCs are vehicles on which a High-Performance Computer (HPC) is mounted. They can go to the location of the User Nodes, pick up their tasks, process them and deliver the results back to the User Node. The premise of the work was the possibility of computation (on the MHPC) while moving. We have modeled this problem as an MILP and have used insights from the model to propose heuristics to solve the problem at scale. An extensive evaluation of various scenarios and effects of different parameters in the system has been performed as well.

- **Message Ferrying with a Purpose (HPC+MF Problem):**This work has considered an algorithm for a similar problem where the communication and computation component of the MHPCs is separated as follows: A stationary HPC is placed somewhere in the field (similar to military base stations) and the vehicles act as "Message Ferries (MF)" that provide communication (moving tasks and results) between the User Nodes and the HPC. In this scenario, the HPC will take the role of the computational component. We have described the fundamental differences of this problem to the MHPC system which necessitates proposing completely new heuristics for the system. We have developed the framework, mathematical formulation, scalable heuristics and their implementation, and evaluation of the system in Chapter 3.

After extensive study of the above problems, in Chapter 4, we have investigated heuristics that instead of planning for the mobility of the vehicles (MHPCS or MFs) use pre-determined routes of vehicles and piggyback computation or communication services on this movement. We have explored this possibility in detail and have presented two more heuristics for these scenarios in MHPC and HPC+MF frameworks. We have then provided extensive comparison of the these systems that are based on a characteristic of "non-controlled mobility" with those of Chapter 2 and 3 that enjoy "controlled mobility". We also compared the MHPC and HPC+MF heuristics in Chapter 4 noting that the former enjoys "computation on the move" while the latter does not.

In Chapter 5 of the thesis, we have made first steps towards the next possible addition to this work, integration of unknown user mobility in the MHPC and HPC+MF frameworks. We have considered methods for using easily collectible contact traces and proposed techniques to infer highly demanded mobility traces from them. This is complemented by detailed description and analysis of the proposed algorithm.

There are many avenues of research that can be followed from this work. These

include:

- **Integration of Unknown User Node Mobility:** This extension of our work includes consideration of the frameworks where user nodes move and various, possibly low, levels of information about their mobility is given into the future. One can expand the MHPC and HPC+MF heuristics for such scenarios. In Section 5.1 we have categorized these avenues.

  For the case that location of each user node is known for any instance in time, we can break down the mobility of each user node into way points. We have shown that one can find the earliest time and location that a vehicle with controlled mobility and fixed speed can intersect a user node with non-controlled but known mobility. This is done in two steps; first by finding the first two way points on the route of the user node that the vehicle can intersect, then by solving a quadratic equation that yields the time and location of the intersection. With this and knowledge of mobility of all user nodes, the extension to integrate mobility seems straightforward. However, the assumption of knowing mobility for all time instants through future is not realistic. A second suggestion is to track the user node until it is met. In this scheme, the vehicle can ask for the current location, direction, and speed of a user node and assume that it moves with those specifications forever and find the intersection point. This intersection point is guaranteed to be found by solving a quadratic equation. After finding the intersection point the vehicle must go to that location. If the user node is present, i.e. it does not change direction by the time of intersect, the exchange of information can happen; otherwise the vehicle must inquire the location, direction, and speed of the user node again and repeat the same procedure from its new location. The problem with this approach is that the number of such attempts must be limited to be able to deliver reasonable service in the system. A final recommendation to accommodate mobility involves

assuming that user nodes update the corresponding vehicles about any change of direction. In this case the vehicle can initially go on an intersect course and whenever it gets an update from the user node, it can change the intersect course accordingly. The issue with this approach is also similarly the problem of chasing user nodes for a long time. In both cases, limitation of the number of attempts seems necessary.

- **Integration of Complex Communication Models:** Another recommended addition is modification or proposal of new heuristics for more complex communication scenarios. These can include scenarios where the short-range and long-range radios of the systems suffer from various imperfection of wireless channels. For example, one can modify the mobility of the MHPCs or MFs to get optimally close to the user nodes for information exchange, considering that the closer they get, the better their experienced communication quality is expected to be. Other considerations that are worthy are integration of various wireless effects like fading or shadowing into the model.

- **Handling Imperfect Data Transmissions:** With an imperfect wireless channel, loss of data is inevitable. In our work, the assumption is that such losses are handled by TCP or another underlying transport protocol. While it is possible to handle the loss at lower levels, it may also be worth considering handling the loss at the application level. This will provide an opportunity to integrate the loss handling into the heuristics directly. For example, one can consider rejecting tasks if the actual task or the result fail to be transferred between a vehicle and the user node after a given number of attempts.

- **Redundancy:** The current implementation of our heuristics assume that once a vehicle is assigned to a task, it will be able to serve it successfully unless the task deadline is missed. However, there may be other reasons resulting in failure

of service to a task including communication imperfections (as noted above), vehicle malfunctions, road conditions, etc. While rejecting tasks assigned to such vehicles is one option, another approach to this issue can be using redundancy. In such cases, for some or all tasks more than one vehicle can be responsible for providing service. In the event that one of the vehicles fails, the others may make up for the failure.

- **Distributed Systems:** Proposing distributed heuristics for the MPHC problem seems another appealing addition. One may consider the case that there is no need for an MHPC controller and suggest mechanisms to coordinate among the MHPCs to process the tasks. This problem can introduce many new issues including proper state synchronization among the vehicles. Request for service in these cases can be sent as a broadcast or multicast message that is possibly received by many vehicles. Once a vehicle commits to service, it shall notify the others of the state of service being provided and if successful, there will not be a need for other vehicles to help serve that task. Redundancy can be added to the distributed solution as well making it capable of serving tasks in the event of failure of one vehicle.

- **Message Relaying:** Similar to the work of Zhao et al. [58], it is also possible to use the user nodes and the vehicles (MHPCs or MFs) as message relays that help in communication and facilitate the transfer of data among the components of the system. This addition can be broken down to the following possibilities: (i) an MHPC may have a task picked up, partially processed or ready for delivery. It can leave that task at a user node that is not supposed to receive the results so that another MHPC that can provide a more timely service can pick it up and continue with the serving of that task. (ii) In a similar scenario as i, an MHPC can meet another MHPC and exchange some of its tasks with it if it

determines that the other MHPC can serve those tasks better. This second scenario needs handling of mobility of the vehicles and might be considered in conjunction with integration of mobility to the model. (iii) An MF that has a task picked up or has a result to be delivered can similarly use any regular node as a relay. Note that our HPC+MF heuristic already uses the special HPC node as a relay as it does not require the pickup MF to be the same as the delivery MF. This possibility can be extended to using regular nodes as relays if it improves the service. (iv) Finally, in the same scenario as iii, MFs that meet each other on their routes or get into each others communication range can exchange their picked up tasks or ready-to-deliver results. This will happen if the other MF will visit the HPC (in case of pickup) or the delivery user node (in case of delivery) sooner.

- **Addition of more Features to the System:** There are many additional features that can be added to the system and may need modification of the heuristics. To name a few, processor sharing in the schedulers, more complex scheduling models, consideration of physical size of the tasks/results and the required time to transfer them among components of the system, and consideration of the computation of the optimal solution as yet another tasks that shall itself be scheduled are all examples of possible additions to our work.

- **Building the Systems:** Finally, a last proposal includes building an actual system that uses vehicles and deployed user nodes and tests the functionality of the proposed heuristics and possibly reveals the the required modifications to deal with real-world situations. This can be done in various scales, from using small computers and controlled robots to deploying the actual system in a battlefield. One can consider to study the problem on a smaller scale using Raspberry Pi [1] computers mounted on robots with controlled mobility. While

this can mimic an MHPC and might reveal insights regarding the problem and issues that may be faced with a full-scale implementation, it will still be far from a full implementation of the system which will use actual military vehicles similar to those of Figure 1a to serve computational tasks of soldiers in a combat field.

REFERENCES

[1] "Raspberry pi foundation." Raspberry Pi 2.

[2] "IBM ILOG CPLEX Optimizer." urlhttp://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, Last 2010.

[3] "CERIT-SC (a part of metacentrum) workload log," tech. rep., CERIT-SC and the Czech National Grid Infrastructure MetaCentrum, 2013.

[4] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M., "Above the clouds: A berkeley view of cloud computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[5] Aschenbruck, N., Gerhards-Padilla, E., and Martini, P., "Modeling mobility in disaster area scenarios," *Performance Evaluation*, vol. 66, no. 12, pp. 773–790, 2009.

[6] Aschenbruck, N., Munjal, A., and Camp, T., "Trace-based mobility modeling for multi-hop wireless networks," *Computer Communications*, vol. 34, no. 6, pp. 704–714, 2011.

[7] Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., and Yang, H.-I., "The case for cyber foraging," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pp. 87–92, ACM, 2002.

[8] Banerjee, N., Corner, M., and Levine, B., "An energy-efficient architecture for DTN throwboxes," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 776–784, IEEE, 2007.

[9] Banerjee, N., Corner, M. D., and Levine, B. N., "Design and Field Experimentation of an Energy-Efficient Architecture for DTN Throwboxes," *IEEE/ACM Transactions on Networking*, vol. 18, pp. 554–567, April 2010.

[10] Bin Tariq, M. M., Ammar, M., and Zegura, E., "Message ferry route design for sparse ad hoc networks with mobile nodes," in *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pp. 37–48, ACM, 2006.

[11] Camp, T., Boleng, J., and Davies, V., "A survey of mobility models for ad hoc network research," *Wireless communications and mobile computing*, vol. 2, no. 5, pp. 483–502, 2002.

[12] Chen, M., Hao, Y., Li, Y., Lai, C.-F., and Wu, D., "On the computation offloading at ad hoc cloudlet: architecture and service modes," *Communications Magazine, IEEE*, vol. 53, no. 6, pp. 18–24, 2015.

[13] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A., "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, pp. 301–314, ACM, 2011.

[14] Colliat, G., "OLAP, relational, and multidimensional database systems," *ACM Sigmod Record*, vol. 25, no. 3, pp. 64–69, 1996.

[15] Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P., "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62, ACM, 2010.

[16] Dinh, H. T., Lee, C., Niyato, D., and Wang, P., "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, 2011.

[17] Eltoweissy, M., Olariu, S., and Younis, M., "Towards autonomous vehicular clouds," in *Ad hoc networks*, pp. 1–16, Springer, 2010.

[18] Fernando, N., Loke, S. W., and Rahayu, W., "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[19] Flinn, J., "Cyber foraging: Bridging mobile and cloud computing," *Synthesis Lectures on Mobile and Pervasive Computing*, vol. 7, no. 2, pp. 1–103, 2012.

[20] Forbes, D., "TSY-300X 3U VPX, 8-Slot Preconfigured System, high performance computers (HPCs)." Themis Computer.

[21] Gao, L., Yu, S., Luan, T. H., and Zhou, W., *Delay Tolerant Networks*. Springer, 2015.

[22] Gerla, M., "Vehicular cloud computing," in *Ad Hoc Networking Workshop (Med-Hoc-Net), 2012 The 11th Annual Mediterranean*, pp. 152–155, IEEE, 2012.

[23] Habak, K., Ammar, M., Harras, K. A., and Zegura, E., "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," *IEEE 8th International Conference on Cloud Computing*, 2015.

[24] Jain, S., Fall, K., and Patra, R., *Routing in a delay tolerant network*, vol. 34. ACM, 2004.

[25] Jain, S., Shah, R. C., Brunette, W., Borriello, G., and Roy, S., "Exploiting mobility for energy efficient data collection in wireless sensor networks," *Mobile Networks and Applications*, vol. 11, no. 3, pp. 327–339, 2006.

[26] Jea, D., Somasundara, A., and Srivastava, M., "Multiple controlled mobile elements (data mules) for data collection in sensor networks," in *Distributed Computing in Sensor Systems*, pp. 244–257, Springer, 2005.

[27] Keränen, A., Ott, J., and Kärkkäinen, T., "The one simulator for dtn protocol evaluation," in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, Simutools '09, (ICST, Brussels, Belgium, Belgium), pp. 55:1–55:10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[28] Lee, K., Hong, S., Kim, S. J., Rhee, I., and Chong, S., "SLAW: A New Mobility Model for Human Walks.," in *INFOCOM*, pp. 855–863, IEEE, 2009.

[29] Lewis, G., Echeverría, S., Simanta, S., Bradshaw, B., and Root, J., "Tactical cloudlets: Moving cloud computing to the edge," in *Military Communications Conference (MILCOM), 2014 IEEE*, pp. 1440–1446, IEEE, 2014.

[30] Li, B., Pei, Y., Wu, H., and Shen, B., "Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds," *The Journal of Supercomputing*, pp. 1–28, 2015.

[31] Lindgren, A., Doria, A., and Schelen, O., "Probabilistic Routing in Intermittently Connected Networks.," in *SAPIR* (Dini, P., Lorenz, P., and de Souza, J. N., eds.), vol. 3126 of *Lecture Notes in Computer Science*, pp. 239–254, Springer, 2004.

[32] Mell, P. and Grance, T., "The nist definition of cloud computing," 2011.

[33] Meroni, P., Gaito, S., Pagani, E., and Rossi, G. P., "CRAWDAD data set unimi/pmtr (v. 2008-12-01)." Downloaded from `http://crawdad.cs.dartmouth.edu/unimi/pmtr`, Dec. 2008.

[34] Monfared, A., Ammar, M., Zegura, E., Doria, D., and Bruno, D., "Computational ferrying: Challenges in deploying a mobile high performance computer," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*, pp. 1–6, IEEE, 2015.

[35] Morris, E., "A new approach for handheld devices in the military," *SEI Blog*, 2011.

[36] Natarajan, A., Motani, M., and Srinivasan, V., "Understanding urban interactions from bluetooth phone contact traces," *Passive and Active Network Measurement*, pp. 115–124, 2007.

[37] Olariu, S., Hristov, T., and Yan, G., "The next paradigm shift: from vehicular networks to vehicular clouds," *Basagni, S. and Conti, M. and Giordano, S. Stojmenovic, I),(Eds), Mobile Ad hoc networking: the cutting edge directions, Wiley and Sons, New York*, 2012.

[38] Rahimi, M. R., Ren, J., Liu, C. H., Vasilakos, A. V., and Venkatasubramanian, N., "Mobile cloud computing: A survey, state of art and future directions," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 133–143, 2014.

[39] Rhee, I., Shin, M., Hong, S., Lee, K., Kim, S. J., and Chong, S., "On the levy-walk nature of human mobility.," *IEEE/ACM Trans. Netw.*, vol. 19, no. 3, pp. 630–643, 2011.

[40] Ristanovic, N., Tran, D., and Le Boudec, J., "Tracking of mobile devices through Bluetooth contacts," in *Proceedings of the ACM CoNEXT Student Workshop*, p. 4, ACM, 2010.

[41] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N., "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[42] Shah, R. C., Roy, S., Jain, S., and Brunette, W., "Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks," *Ad Hoc Networks*, vol. 1, no. 2, pp. 215–233, 2003.

[43] Shi, C., Ammar, M. H., Zegura, E. W., and Naik, M., "Computing in cirrus clouds: the challenge of intermittent connectivity," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 23–28, ACM, 2012.

[44] Shi, C., Habak, K., Pandurangan, P., Ammar, M., Naik, M., and Zegura, E., "Cosmos: computation offloading as a service for mobile devices," in *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pp. 287–296, ACM, 2014.

[45] Shi, C., Lakafosis, V., Ammar, M. H., and Zegura, E. W., "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, pp. 145–154, ACM, 2012.

[46] Shires, D., Henz, B., Park, S., and Clarke, J., "Cloudlet seeding: Spatial deployment for high performance tactical clouds," *Parallel and Distributed Processing Techniques and Applications*, 2012.

[47] SIGFOX, "White paper: M2m and iot redefined through cost effective and energy optimized connectivity," tech. rep., 425, rue Jean Rostand, 31670 Labege FRANCE, 2015.

[48] Soyata, T., *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*. IGI Global, 2015.

[49] Srinivasan, V., Motani, M., and Ooi, W. T., "CRAWDAD data set nus/contact (v. 2006-08-01)." Downloaded from `http://crawdad.cs.dartmouth.edu/nus/contact`, Aug. 2006.

[50] Teller, P., McGarry, M., Shires, D., Park, S.-J., Naguyen, L., and Deroba, J., "Enabling battlefield decision making in the tactical cloud," in *Army High Performance Computing Research Center*, 2013.

[51] Wang, P., Gao, Z., Xu, X., Zhou, Y., Zhu, H., and Zhu, K., "Automatic inference of movements from contact histories," in *Proceedings of the ACM SIGCOMM 2011 conference on SIGCOMM*, pp. 386–387, ACM, 2011.

[52] Whaiduzzaman, M., Sookhak, M., Gani, A., and Buyya, R., "A survey on vehicular cloud computing," *Journal of Network and Computer Applications*, vol. 40, pp. 325–344, 2014.

[53] Whitbeck, J., Conan, V., and Dias de Amorim, M., "Critical analysis of encounter traces," in *Proceedings of the 2010 ACM workshop on Wireless of the students, by the students, for the students*, pp. 29–32, ACM, 2010.

[54] Whitbeck, J., De Amorim, M., Conan, V., Ammar, M., and Zegura, E., "From encounters to plausible mobility," *Pervasive and Mobile Computing*, vol. 7, no. 2, pp. 206–222, 2011.

[55] Yoon, J., Liu, M., and Noble, B., "Random Waypoint Considered Harmful.," in *INFOCOM*, 2003.

[56] Zhang, Y., Niyato, D., Wang, P., and Tham, C.-K., "Dynamic offloading algorithm in intermittently connected mobile cloudlet systems," in *Communications (ICC), 2014 IEEE International Conference on*, pp. 4190–4195, IEEE, 2014.

[57] Zhao, W., Ammar, M., and Zegura, E., "A message ferrying approach for data delivery in sparse mobile ad hoc networks," in *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pp. 187–198, ACM, 2004.

[58] Zhao, W., Ammar, M., and Zegura, E., "Controlling the mobility of multiple data transport ferries in a delay-tolerant network," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2, pp. 1407–1418, IEEE, 2005.

[59] Zhao, W., Chen, Y., Ammar, M., Corner, M., Levine, B., and Zegura, E., "Capacity enhancement using throwboxes in dtns," in *Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference on*, pp. 31–40, IEEE, 2006.