Framework Design and Job Scheduling for Vehicular Cloud Computing

FENG JINGYUN

Doctor of Philosophy

Department of Informatics
School of Multidisciplinary Sciences
SOKENDAI (The Graduate University for Advanced Studies)

Framework Design and Job Scheduling for Vehicular Cloud Computing

Feng Jingyun

Doctor of Philosophy

Department of Informatics

School of Multidisciplinary Sciences

SOKENDAI (The Graduate University for Advanced Studies)

Contents

List of Figures

List of Tables

Abstract

Vehicles nowadays are not just tools transporting people from one place to another. Drivers or passengers usually demand more features to be provided on the road, like driving assistance, multimedia services and more.

The inherent problem is how to meet this ever-increasing computational demand in vehicles. Usually, more features mean more hardware to be installed, along with higher cost. However, the emergence of cloud computing and edge computing technologies change this. By using a pool of resource to provide flexible services through network, users are freed from upgrading and maintaining hardware. How to apply such technologies in vehicular environment becomes a problem.

At first, we proposed a framework to allow vehicles pool their computation resources and help each other in processing. Different from the existing solutions, this framework does not rely on deployed infrastructures, which are not available everywhere at the moment, but utilizes the new DSRC standard and a decentralized algorithm to distribute computation tasks. It can be the fundamental of a more generic and scalable vehicular cloud computing framework.

In the framework, the software system is divided into 3 main components: application, backend and manager. And then we propose a mechanism to cache multiple jobs for optimal assignment and an algorithm to solve the assignment problem. The framework also takes the predictable movement pattern in vehicular environment into account.

A new Ant Colony Optimization (ACO) based algorithm, with a polynomial time complexity, is designed. Preliminary evaluation shows the superiority of the algorithm comparing to brute force searching and random assignment.

Evaluation environment is built with *Omnet++* network simulator, *Simulation of Urban MObility (SUMO)* traffic simulator and *Veins* simulation framework. Then realistic traffic scenario, *Luxembourg SUMO Traffic scenario*(LuST), with 24 hours traffic from real-world traffic demand data, in the Luxembourg City, Luxembourg is used. The evaluation is done on both the urban part and highway part

of this scenario, each with a 15 minutes time span. Results from repeated experiment indicate the proposed framework outperforms competing schemes in both environments, and achieving more than 90% optimal assigning solutions.

And then, we consider the scenario with already deployed infrastructures, like roadside units and cellular base stations. Hence the new framework, *Hybrid Vehicular Cloud* (HVC) is designed. HVC aims at the utilization of edge resources, like roadside units and idle neighbor vehicles, as augmentation of the remote cloud to provide computational services. With DSRC connections to roadside units and other vehicles, HVC also reduces communication costs in offloading with cellular network. To better utilizes the dynamic resources on road, we use the "available time" concept to model the connection status change with movements of vehicles. A handshaking procedure is introduced to eliminate offloading failure due to volatile connections in vehicular network. An online assignment algorithm is proposed to assign jobs in a heuristic way that as many as possible jobs are guaranteed to be finished in time, while still leaving time slots for jobs arriving in future.

Simulation is done with the aforementioned LuST scenario. Two roadside units are set in the traffic map and an ideal assumption on cellular network is made. It is found that the new proposal achieves 100% job finishing ratio in almost all settings, while reducing more than 50% cellular traffics in most cases.

In addition to the two proposed framework, we then discuss the extensions to them and alternative application scenarios. We also make some anticipations on the future development in this field.

Chapter 1

Background

1.1 Vehicular applications and challenges

Vehicle nowadays is not just a mechanic tool transporting people from on place to another, but also becoming part of our digital life. As studied by *AAA Foundation* indicated, active drivers spend over 70 minutes on the road per day[1]. Therefore, higher quality of safety and convenience services are demanded. For example, drivers would want a more intelligent navigating system that base not only on distance, but with smart congestion avoidance, on-demand path choosing (e.g. traveling past some demanded point if feasible), parking slot scheduling, etc. Also recently proposed head-up displays on vehicles can provide drivers more information during driving, without distracting them from road.

As recent emerging trend of self-driving vehicles[2], which frees the driver from concentrated driving labor, and on-car multimedia services like *Apple CarPlay* [3], an increasing number of entertainment-related computer applications are expected in vehicular environments. Some example applications are like games for passengers or the driver who is free from driving and want to have some fun on the long and boring road, and multiview applications to provide surrounding sceneries to users in the center of crowded traffic.

This brings a new challenge, i.e. how to satisfy various types of job execution demands from

Table 1.1: Vehicle applications

	Examples	Priority	Offload
Crucial Applications (CAs)	Vehicle control, system monitoring, accident prevention	Highest	No
High Priority Applications (HPAs)	Navigation, information services, optional safety applications	High	Yes
Low Priority Applications (LPAs)	Multimedia, passenger entertainment	Medium to Low	Yes

applications. Some of these applications require high computational capabilities, such as speech/image recognition, self-driving, and sensor data processing. And as it is like in mobile phones, there is never a device can promise satisfaction to every application. Upgrading the on-board computers is one option, but the cost is high. Cloud computing [4] is another possible solution, which uses resources hosted on the Internet to manage and process jobs.

To solve this problem, we first take an investigation into types of vehicular applications.

Vehicle applications can be classified into three levels according to their characteristics: *crucial applications* (CAs), high-priority applications (HPAs) and low-priority applications (LPAs), as shown in Table 1.1. CAs are core applications of the vehicle system or safety-related applications. Because of their importance to the vehicle and the passengers, CAs have the highest priority and must be flawlessly executed without relying on unstable connections in the vehicular environment. Moreover, because CAs are generally developed by the vehicle manufacturer, the vehicle's on-board system should always be designed to have sufficient–if not abundant–capacity to meet their resource demands. Hence, CAs are considered to be executed completely outside of AVE.

The remaining applications are divided into high-priority applications (HPAs) and low-priority applications (LPAs) depending on their purposes. HPAs include driving-related applications and optional safety-enhancing applications, e.g., routing and information services for drivers. These applications

are important but not obligatory, which means that failures and delays are allowed, although inconvenience will be caused to the driver. Typical HPAs include heads-up displays (HUDs)[5], field-of-view enhancement[6], and road sensing[7]. An increasing number of new cars are equipped with such applications. To accommodate these emerging HPA services, manufacturers are designing some computational capacity margins into their on-board systems.

LPAs are the remaining class of applications that are of lesser importance to drivers and passengers. For example, speech recognition[8] is an interesting application that allows a driver to issue various commands without being distracted from driving. This concept has been adopted in Apple's CarPlay[3], whose implementation relies on an attached smartphone and cloud computing via the cellular network. Other multimedia applications, such as video processing[9], have also been proposed for vehicles. With the emerging trend of self-driving vehicles, users are becoming able to shift their focus from driving to other activities, such as entertainment. Cloud-based video games would provide a better travel experience for passengers or a free driver. With the further development of smart vehicles, an increasing number of HPAs and LPAs will emerge. Note that HPAs and LPAs are interchangeable; i.e., HPAs can be LPAs for some vehicles, and LPAs can be HPAs for some vehicles. We use these two terms to indicate that applications other than CAs may also differ in their importance to users.

The inherent problem is how to meet the ever-increasing computational demands posed by LPAs and HPAs in vehicles. The demand is always increasing but the hardware does not upgrade themselves. The simplest answer is to buy a new, more powerful car when more applications are required. However, a new vehicle is extremely expensive. Also there are few people who would just change the cars for one or two applications they want to run in their cars. Another intuitive way is like nowadays desktop computers, we can standardize the on-board computer components so that they can be easily replaced and upgraded. However, because the confidentiality of vehicle system designing in the current state of car industry and the fact that on-board system is not the main part of a vehicle, it is not likely to reach this goal. This situation can repress the development of vehicular applications.

Considering these, instead of replacing equipment or even the entire vehicle, we propose a solution based on the efficient utilization of the existing computational resources can ease the burden of the

deployment of new vehicular applications.

1.2 Cloud computing methodologies

1.2.1 Cloud computing

In recent years, cloud computing has emerged as a solution for problems similar to those discussed above in environments with wired connections[10]. By offloading the local workload to a remote cloud, the owner of a device can be free from the need to upgrade and maintain additional hardware[11]. This concept was quickly transplanted into the mobile environment, where it is also known as mobile cloud computing (MCC)[12]. MCC provides a good solution for mobile devices, which typically exhibit poor performance in terms of computational capability and require a high cost to upgrade.

However, in the mobile scenario, offloading faces challenges such as unstable network connections, high latency and network data charges. Some of these problems are even more severe in vehicular environments. The mobility of vehicles is considerably higher than that of hand-held devices, which significantly reduces the link durations with fixed-point infrastructures such as roadside units (RSUs)[13][14]. In addition, unlike hand-held devices, which mostly roam in urban areas, vehicles are often in rural areas, where infrastructures are rarely deployed and Internet connections to the remote cloud are not as good as those in urban areas. Because of these limitations, cloud computing is difficult to use in vehicular environments.

The problems brings difficulty in realizing vehicular applications[15]. For example, *augmented reality* (AR) is proposed to provide helpful information and warnings in a *heads-up display* (HUD) for vehicles[5] or to provide a better field of view[6], and more[16][17]. However, AR requires rapid processing, which is unrealistic using traditional cloud services due to the high latency. Hence, directly applying existing cloud computing frameworks on vehicles is not suitable.

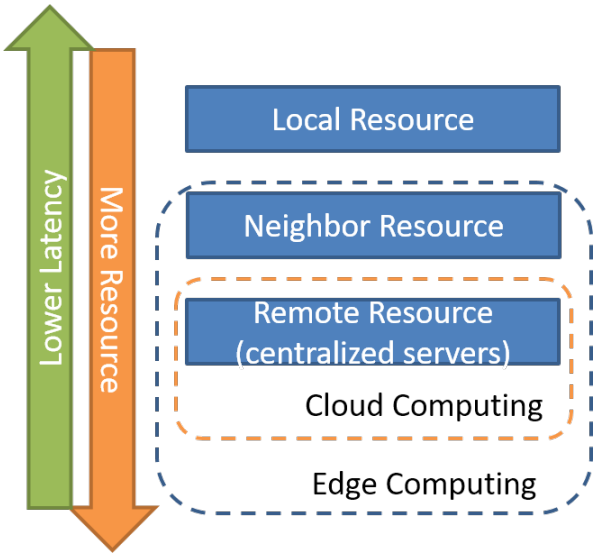


Figure 1.1: Cloud Computing, Edge Computing and the resources they utilize.

1.2.2 Edge computing

Mobile edge computing (MEC)[18][19] is another option for addressing these issues. This concept involves placing computational resources at the logical extremes of the network, e.g., access points, RSUs, base stations, or even users' devices, instead of relying only on a centralized cloud. In this way, MEC enables applications to respond in real time.

The main difference that distinguishes cloud and edge computing from other methodologies is the ability to "virtualize" resources, meaning that resources are provided not in physical forms (such as CPUs and HDDs) but rather in a per-time and per-use manner. These methodologies allow more flexible services to be provided with lower upgrade and upkeep costs to consumers. The resources those methodologies utilized are illustrated in Fig. 1.1.

The aspect of MEC that interests us the most is the idea of using users' devices as computing nodes, i.e., pooling the available resources to construct a local "cloud". By sharing resources among participating nodes, the necessity for deployed infrastructures or connections to a centralized server is eliminated. The next question is whether this type of computational resource sharing is possible in vehicular environments. Our answer is positive, based on the following properties:

1. *Shareable resources*: As previously mentioned, apart from the computational resources used by CAs, some resources remain available for HPAs/LPAs. When these applications are not running (e.g., navigation applications become idle after completing the computation of routes), this subset of resources is in an idle state and can be shared.

2. *Movement pattern*: Although vehicles have a high speed relative to stationary infrastructures, with which their link durations are short, the speed difference between two vehicles driving in the same direction on the same road is much smaller[20]; hence, a longer link duration between such vehicles becomes possible.

3. *Energy*: A formidable challenge in MEC is the limited battery life of mobile devices. Assisting others is not "free" because it consumes the host device's energy. By contrast, a vehicle has much larger battery storage, and the on-board system's energy cost is trivial compared with the total energy consumed during driving.

4. *Incentives*: Engaging in such a resource-sharing framework is beneficial to both service providers and users. Service providers can reduce their infrastructure deployment costs. Users can potentially enjoy low-latency computational services at the cost of nothing but idle resources. Moreover, service providers may even pay users for their shared resources, which can be cheaper than deploy service enabling infrastructures.

Many unresolved issues remain. Challenges exist with regard to framework design, the method of self-organization, and scheduling in the distributed environment. Studies have previously been conducted on resource sharing in vehicular environments, such as [21][22]. However, to the best of the authors' knowledge, no solution has been proposed for general applications, nor one that would operate in a completely distributed manner.

1.2.3 Offloading and virtualization

There are two key characteristics in cloud computing and edge computing methodologies: offloading and virtualization.

Offloading is the process of migrating workloads to other machines for processing; we define these workloads as *jobs* in our work. Offloading is one fundamental aspect of the proposed framework. It can be performed at various levels, such as functions, tasks, applications and virtual machines (VMs) [23]. For example, a navigation application may offload only a few parameters like "source: (x_1, y_1), destination: (x_2, y_2)", and the corresponding backend software maps the coordinates into its map database and runs routing algorithm; an image recognition application may offload photo data and some instructions to be operated on the photo; a scientific application may offload a function and its input as a workload unit; a multithread application may offload one of its threads's code and memory data, and makes it compute on another machine. We call all of those as "jobs", despite their large differences in scale and size.

A considerable amount of work has been performed on offloading itself. One typical proposal is *CloneCloud*[24]. *CloneCloud* focuses on how to split the execution of workloads and move them to another machine. This framework borrows this idea and will not reinvent the wheel. Rather, we will focus on processor discovery and job scheduling. Although the workload concept is generalized, the following factors are considered:

1. *Size of the transmitted data.* The network bandwidth is limited in a vehicular environment; hence, the amount of data that needs to be transmitted is one critical factor regarding performance.

2. *Host requirements.* Some jobs require specific software or capable hardware. In traditional cloud computing, this is not a problem because cloud servers can always satisfy these requirements. However, these requirements may not be satisfied for some nodes.

3. *Importance.* Not all jobs are of equal benefit to users. Some jobs (e.g., jobs associated with HPAs) have higher priority, and the greater effort should be made to complete them.

Virtualization is the abstraction of computational resources, typically as virtual machines (VMs) with associated storage and networking connectivity. For better virtualization (i.e., better utilization of the available computational resources), the processing of an offloaded job is performed in a VM on a processing node. This VM has two functionalities: first, it includes the required support for the

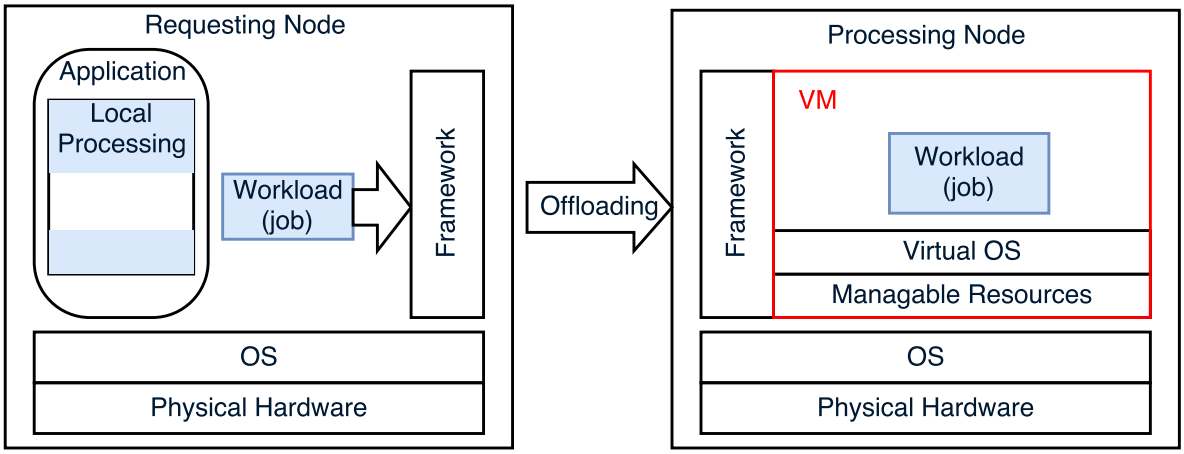


Figure 1.2: Illustration of the offloading process. The requesting node splits a workload and submits it to our framework. The framework offloads the job to a processing node, whose framework sets up a VM to process it.

workload, e.g., operating system (OS) and runtime libraries; second, it virtualizes and manages the remaining resources, excluding those occupied by CAs, so that the execution of CAs is isolated from that of LPAs/HPAs and the priority of the CAs can be guaranteed. The virtualized resources of this VM are what will be scheduled in our proposed framework. The aforementioned offloading process is illustrated in Figure 1.2.

1.3 Vehicle systems

1.3.1 Hardware

Modern vehicles contain many electronic components, such as sensors, valve controllers, diagnostic systems, etc. Components are connected with a bus-based system, and communicates with specific protocols, that focus on real-time and accuracy [25]. Controller Area Network (CAN) [26] is one of those protocols that are widely adopted.

A vehicle also has a microcomputer to manage other components. On a traditional vehicle that is usually an 8-bit microcontroller on a chip, with a ROM of multiple kilobytes and RAM under 1 kilobyte. For typical operations it is sufficient, but definitely not able to support a smart vehicle.

In recent years, as demand of multimedia and intelligent systems arise, in-vehicle computers have been improved. Some in-vehicle computers carries Intel Core processors that have 2 cores and $2\,GHz$ clock rate. Autonomous cars that utilize deep learning requires higher processing power, e.g. Nvidia's in-vehicle computer *Drive PX2* [27] is reported to have 6 cores CPU, GPU and up to $16\,GB$ RAM, supporting 320 TOPS computing.

Seeing these, in this thesis we consider vehicles' processing power is somewhere between a smartphone and a desktop computer. That will allow scheduling and some of the processing to be done locally, as well as support for the proposed VM.

1.3.2 Software

Like hardware, software in vehicles varies. Popular operating systems (OS) adopted by car manufacturers include Windows (Windows Embedded Automotive), Linux (Genivi, etc.) and a real-time OS named QNX. Manufacturers usually do customizations on the systems, or develop new ones upon existing systems. This makes the market quite scattered. However, as in recent years IT companies like Google and Baidu enter the auto industry, traditional manufacturers are driven to separate software and hardware and provide more flexibilities[28]. We expect software in vehicles will become more open and standardized in near future.

Those general purpose OS's have the potential to support various applications, allowing third party developers to make their own software, and also allowing our proposed framework to run on them. Nevertheless, we try to avoid dependency on platforms, like library and OS specific features, in our frameworks. So that they can apply on whatever systems that will be on future smart vehicles. But note that applications and their jobs may still be limited by platforms, which is considered in our proposals.

1.4 Related work

Researches have been done for resources, such as computation, storage, and network resources, in cloud computing to be virtualized to provide flexible services rather than offering exclusive infrastructures to users. By using cloud computing, users can have a considerably larger available resource pool to cover surges (fluctuating utilization patterns) while paying less upkeep costs for short-term and low-demand usage [4].

An increasing number of applications are being used on mobile devices, and this presents a new challenge for such devices. The reason is that because of their embedded hardware, mobile devices are subject to limitations in terms of their computational capabilities and battery life, among others. Mobile cloud computing (MCC)[29], the technology of augmenting mobile devices with a cloud or even creating a cloud within them, has recently emerged as a hot topic. By applying cloud computing on mobile devices, the computation capability can be substantially improved [12]. Currently, there are some existing mobile cloud computing frameworks, such as *MAUI* [30], *MobiCloud* [31], and *CloneCloud* [24]. Moreover, *Yang* et al. [32][33] proposed offloading schemes for applications that require low latency in MCC. The main concerns in these works are battery life, connection status and execution time.

Edge computing has been researched as a means of augmenting cloud computing. As studied by Shi *et al.*[34], this technology has the potential to shorten response times, increase processing efficiency and reduce network pressure. Ha *et al.*[35] have proposed an architecture for offloading Google Glass workloads to nearby *cloudlets*.

With regard to vehicular environments, several studies have investigated specific applications utilizing cloud resources. Several related studies exist concerning cloud computing systems for vehicular environments, such as *Datacenter at the Airport* [36], *Pics-on-wheels* [37] and *Cloud Transportation System (CTS)* [38]. *Datacenter at the Airport* [36] is an attempt to utilize the resources of idle vehicles to provide cloud services in parking lots. It addresses a relatively static scenario (compared with that of vehicles on the road) and relies on access point devices for organization. Su *et al.*[39] also proposed a framework for utilizing parked vehicles as content caching nodes. Comparing with the traditional

method of content-centric networking (CCN), which relies on RSUs, this proposal imposes less of a burden on RSUs, provides higher capacity and also allows vehicles to communicate via decentralized D2D communication. Similar research was done in [40], in which detailed traffic and network models were built and a game-theory-based method was applied for utility optimization. *Pics-on-wheels* [37] is a proposal in which vehicles are utilized as mobile cameras and organized by a centralized server to provide photo services, where vehicles serve as the sensors for acquiring pictures. The data are uploaded to cloud servers via a cellular network such that users can retrieve information from the cloud. Conversely, the *Cloud Transportation System (CTS)* [38] focuses on a crowd-sourcing scheme for collecting user data and processing them in the cloud for traffic model construction and congestion prediction. Other researches include [41][42][43][44][45]. There are two main differences between AVE and the aforementioned works: the framework we propose is for generic computation offloading rather than any specific application, and the main resources that we utilize are located on moving vehicles.

The direct use of traditional cloud services on the road is unrealistic for some applications, such as AR, for reasons such as high latency. However, edge computing is feasible in vehicular environments because communication between vehicles is constantly improving [46]. Work has been done on utilizing such kind of communications to provide services[47]. Also not all vehicles are running computationally intensive applications at all times. Hou *et al.* have reviewed the problem and possible solutions in [21].

In contrast to these previous works, our aim is to design a framework that is universal to all applications, that fulfills a job's real-time requirement and host requirement (*i.e.*, prerequisite for a job to be performed on the host), and that is adaptable to the vehicular environment. Moreover, an online algorithm is proposed, which immediately schedules the coming computing tasks.

1.5 Structure of thesis

The remaining of thesis is constructed as follows:

In Chapter 2 we introduce our first proposed framework, Autonomous Vehicular Edge Computing Framework (AVE), for use in environments without infrastructures deployed. In the chapter, we first present the application scenario and the problem we face in Section 2.1; then the framework is

introduced in Section 2.2; we also show our proposed workflow to be used in the dynamic environment in Section 2.3 and an ACO-based algorithm to solve the scheduling problem in Section 2.4; finally, we evaluate the performance of this proposal with simulations in Section 2.5.

In Chapter 3 we introduce our next framework, Hybrid Cloud Computing Framework (HVC), which is designed under the assumption that infrastructure and remote cloud access is available. In this chapter, Section 3.2 provides an overview of the framework; Section 3.3 presents the workflow and mathematically formulates the embedded assignment problem; Section 3.4 explains the proposed algorithm for solving the assignment problem; The framework is evaluated by simulations, as described in Section 3.5; In Section 3.6, extensive simulations are done to find out the limitations in VANET and performance with better communication standards.

And then in Chapter 4, some issues presented in network simulation, the contribution and possible extensions of this work, including the application of our scheduling algorithms, the use in generic mobile cloud computing scenarios and extension with centralized management. Finally in Section 4.4.1 and Section 4.4.2, we add some anticipations of cloud computing in vehicular and IoT environment.

In the end of each chapter, except for Conclusion, a brief summary is given.

Chapter 2

AVE: Autonomous Vehicular Edge Computing Framework

2.1 Problems and goals

As the first step, we consider a generalized framework to fill the gap in vehicular cloud computing. The framework is set in an environment lacking infrastructures, and the vehicles try to unite with each other to form an edge network, solving the aforementioned computational resource problems. This problem and the goals of such framework is detailed below.

2.1.1 Application scenario

This framework is considered applied in a decentralized scenario. The scenario is much like current state of vehicular network, in which less infrastructures are deployed, and there is not a centralized service provider to provide cloud computing on the road. But still, there are vehicular applications, whose developers want to extend their feature and whose users want better services. However, for a single application, it is really hard to take advantage of nearby resources without centralized control. That is because unless the application is extremely popular, the probability that nearby nodes are

using it can be very low.

Therefore, a generic cloud computing framework is demanded. The framework is installed along with the vehicle's system, so that vehicles would have it despite what applications the users choose. The framework also has to adapt to the dynamic environment without centralized management.

Then, as we discussed in Section 1.1, applications in the vehicular environment have different priorities. Even in the same application, different jobs also give different gain upon finishing. For example, a navigating application obviously has higher priority for the driver than a music player, and in a navigating application, path finding is clearly more important than nearby hot spots recommendation. Therefore the framework needs to take such difference of jobs into consideration when doing scheduling.

Finally unlike cloud servers, which can be taken as omnipotent for any jobs, vehicles have limited capability, hence not all jobs can be executed on all vehicles. The framework should consider this requirement when doing scheduling.

2.1.2 Goals and requirements

The goal of this framework is to provide universal compatibility to cloud applications, which grants that as many as possible application can be deployed on it without too much concern on underlying details of vehicular environment and get satisfying outcome. Given the various demands of the applications, the following criteria will be considered in this framework:

(i) The ability to finish important jobs faster. As aforementioned, jobs are with priorities. Therefore the framework should be able to prioritize important jobs and finish them as fast as possible. This criteria is represented as the total utility gain from all jobs.

(ii) Delay of finishing jobs. Naturally, we want jobs to be finished as quickly as possible. Hence the delay from job generation to its finish is a key factor to consider in this framework. Considering jobs have different length of workloads, the delay should be normalized with its length so longer jobs are comparable with smaller jobs.

(iii) Overhead of scheduling. While a complete brute force method can find the optimal scheduling

solution for jobs, the time taken in scheduling makes the gain not worth the cost. Therefore, an efficient algorithm that gives satisfiable result is demanded.

(iv) Performance in various environments. Movement patterns of vehicles differs largely between urban streets and highways. The framework, considering its universality, should be applicable and well performing on both scenarios.

2.2 System overview

2.2.1 Roles of nodes

We refer to each vehicle participating in the framework as a *node*. Each job is generated by one node and processed by one node. These two nodes can be identical; i.e., that node that generates the job may also process the job. In this case, we say that the job is processed *locally*. If the node that generated the job cannot directly reach the node that is helping to process the job, then other nodes will be needed to help forward the data between them. For convenience, these three types of nodes are called *requesters*, *processors* and *forwarders*, respectively, in this thesis.

Note that this classification is not permanent and only applies for a particular job. A vehicle may be the processor for one job while also being the requester for another job that it does not have sufficient capacity to process. Furthermore, note that incentive analysis of the vehicles based on game theory, like that in [48], is beyond the scope of this thesis; we assume that vehicles are cooperative and will do no "evil" in this framework.

2.2.2 Software architectures

Like other client-server architectures[24][30], the software in this framework includes client-side applications and corresponding server-side applications, referred as *application* modules and *backend* modules, as shown in Fig. 2.1. *Applications* are run on the native operating system that manages their priorities and available resources; *backends* are run in a VM, that is managed by the AVE framework. We introduce a *manager* module (highlighted in Fig. 2.1) as a middleware to intercept information,

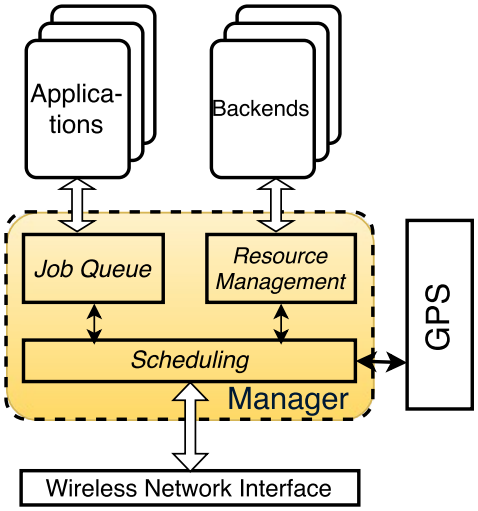


Figure 2.1: Illustration of the software architecture.

jobs (as explained in Section 2.2.3) and results from the other two modules and to make the offloading and job assignment decisions.

Components of this framework are deployed on each vehicle before that vehicle begins traveling on the road. The *manager* module is pre-installed on each vehicle, preferably by the on-board system vendor. *Applications* are chosen and installed by users. A typical means of doing so is to install the applications from the Internet; however, in what way the applications are installed is beyond the scope of our work. *Backends* can be installed together with the corresponding applications (e.g., a user may want an application that can be run *locally* on the vehicle) or deployed independently (e.g., companies may deploy backends on users' vehicles to provide accessibility for their services on the road).

The *manager* module consists of three submodules: a *job queue* module, a *resource management* module and a *scheduling* module. The *job queue* module provides a job offloading interface for applications, collects jobs and places them in the scheduler when the requirements (discussed in Section 2.3) are satisfied. The *resource management* module controls the available resources that the *backends* can use. Note that CA tasks such as safety, monitoring, and emergency reporting are not offloaded and are processed locally with the highest priority. Apart from the resources occupied by CAs, the remaining resources are available to the *backends*. The *scheduling* module is the core of the man-

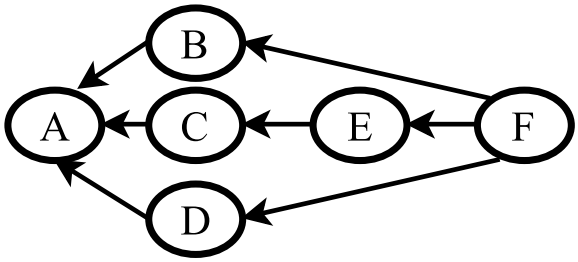


Figure 2.2: Illustration of dependent jobs. A to F are jobs. Arrows indicate that the source relies on the result from the destination. The assumptions adopted here suggest that all jobs from A to F should be either merged into a large independent job or offloaded in order of their hierarchy.

ager. The *scheduling* module is responsible for communicating with other nodes, making assignment decisions, sending jobs and receiving results. In the proposed framework, a GPS (Global Positioning System) device is also required for the scheduling module to obtain the current geographical position and velocity of the vehicle. This requirement is trivial since GPS devices are widely installed.

The network established in our framework uses *dedicated short-range communication* (DSRC) standards, such as *IEEE* 802.11p. This type of communication enables rapid link setup without the establishment of a basic service set. This reduces the overhead incurred for sending data to other vehicles.

2.2.3 Jobs

Each offloaded workload is called a *job*. As previously mentioned, offloaded workloads may occur at various levels, such as functions, applications and virtual machines. To accord with this diversity, jobs are considered here as generalized entities. However, for the modeling of real-life workloads for scheduling, the following job characteristics are assumed in our context:

(i) *Context-free*. This is the basic requirement for a job to be offloaded and executed on another machine. The resource files on which jobs rely are either distributed in the *backends* before offloading or regarded as part of the jobs' data and sent during the offloading period. This characteristic implies dependency of jobs should be handled within the *applications*, before offloading to *manager* module. The implementation that the *applications* use to separate the jobs is out

of our scope. But we suggest that: For jobs that depend on another job's completion, the application can choose to offload them one by one, each after the previous one's result is returned, or to pack all sequential jobs into one larger, independent job. An example is shown in Fig. 2.2: for jobs that depend on each other in this way, the application should either (a) take all inputs for job A to F, merge them into one large job, and then offload them as a whole, or (b) offload A first, then offload B to D when the result of A returns, then offload E afterward, and finally offload F. Hence, regardless of the approach selected, there will be no dependency among the cached jobs, and each job can be scheduled individually. For example, in the *CloneCloud* case, each job is a suspended thread, with all of its related memory data transferred to the processing host.

(ii) *Utility.* Different jobs generally have different impacts on the user experience that differ with their completion times. To represent this, a *utility* factor is considered for each job. Each job carries a utility function that maps its completion time to a cumulative real-number value called the *utility*. The utility also incorporates the importance of the job.

(iii) *Host-specified.* We assume that each node does not necessarily possess all functions necessary to able to process all jobs. The requirements for the processing host are also among the factors considered in the proposed framework.

(iv) *Brief.* As in a distributed environment, each vehicle must provide potential processors with some information about its requested jobs for processing time estimation and further scheduling. The transmission of an entire job in such a query is unrealistic in the bandwidth-limited vehicular environment. Instead, the specification of a small number of parameters, which we call a *brief*, is required from each application to describe a job's workload. The corresponding backend should have the capability to understand the brief and tell how long it needs to process the job.

For the *brief* part, while the content and format of these parameters are determined by the applications and corresponding backends, we suggest an implementation of it: A job's workload is represented by the unit's demanded execution time and its host requirement is represented with *tags*.

Here, we use a simple model to estimate the execution time by comparing the computation capability in terms of CPU speed: assuming that a reference node with a 3.0 GHz CPU takes T seconds to process the job, a node with a 1.5 GHz CPU will take approximately $2T$ seconds. Here, we call the time taken on the reference node the "length" of the job. According to the relative processing speed of a node to the reference node, the processing time can be calculated as *length/relative_rate*.

And then, to represent each job's host requirement, we use identifiers named *tags*. Each tag is denoted by a short series of numbers or symbols (e.g., *"app1"*), representing what is required to process the job. The manager module on each node also holds a set of matching *tags* (e.g., { *"app1", "app2"*}), representing what the node can do. A job j with tag t_j can be processed by node i with set of tags S_i if and only if $t_j \in S_i$.

2.2.4 Neighbor Availability Index (NAI)

To help vehicles make decisions in a distributed manner, a value called the *Neighbor Availability Index* (NAI) is introduced. The NAI can be used to indicate how many potential assisting vehicles are within the reachable area of a requester. Although their available resources may vary, this can still serve as a rough estimate of the nearby resources. Each vehicle calculates its NAI as

$$NAI = \sum_{k=1}^{2} (\text{number of } k\text{-hop idle neighbors}) \cdot \phi^k$$

Here, "idle" means that the neighbor's backend VM is not processing any job using the available resources that it manages. k specifies the number of hops to a neighbor. Because communication via a relay of more than 2 hops is unstable according to existing research [49][50][51], nodes at more than 2 hops are ignored in both the NAI calculation and further scheduling. ϕ is the factor that expresses the fading of resource usability with an increasing number of hops. "Fading" here refers to the fact that as the number of hops increases, the transmission overhead and the probability of losing the connection increases, which results in less gain from offloading. As tested in our simulation scenarios, $\phi = 0.8$ is appropriate, but the most suitable value might differ among different practical cases.

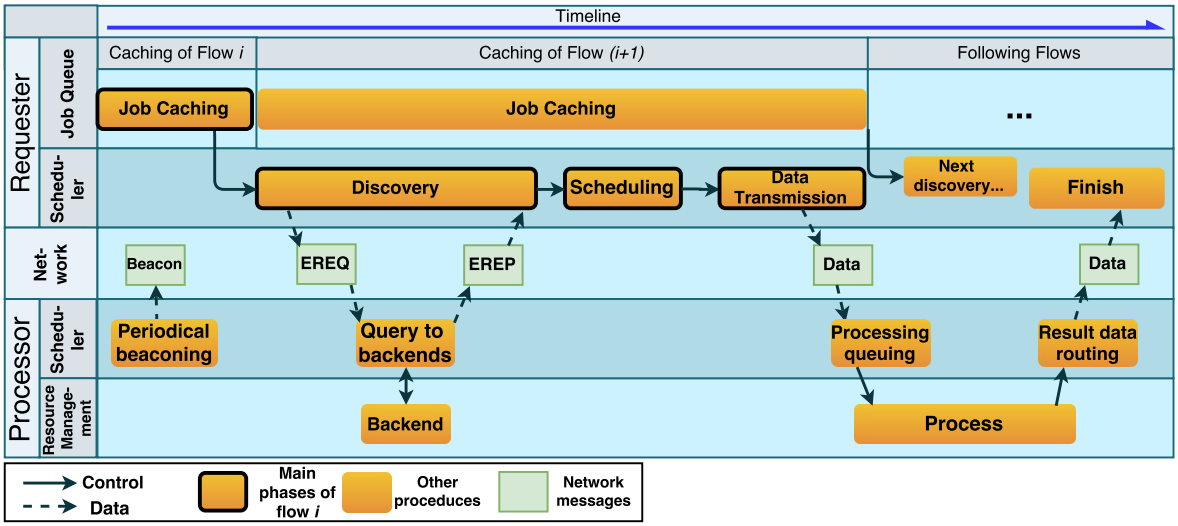


Figure 2.3: Workflow of AVE. The orange boxes with borders correspond to the four main phases discussed in the following sections. Other minor procedures discussed in those sections are also shown.

For calculation of the NAI when needed, AVE establishes an NAI table on each vehicle to store the required information. The entries in the table are in the format $\langle$*vehicle ID, idle state, number of hops, expire time*$\rangle$, and an entry is automatically removed when its *expire time* is reached. The procedure for updating this table is further discussed in Section 2.3.

2.3 Workflow

The workflow of the AVE framework consists of two components: proactive periodic beaconing and reactive *flows*. A *flow* represents the procedures for handling a job from its arrival to the return of the processing results. In this framework, we divide a *flow* into four main phases: *job caching*, *discovery*, *scheduling* and *data transmission*, as shown in Fig. 2.3. The details of beaconing and each phase of a *flow* are explained in the remainder of this section.

2.3.1 Beaconing

The AVE framework employs beacon messages to help vehicles acquire information about the resources available nearby. Each vehicle individually broadcasts a short beacon message at configurable intervals (the default value used in our evaluation is 10 seconds). To reduce network congestion and chance of broadcast storm[52], these beacon messages are not relayed. An AVE beacon message carries the following information:

(i) The source vehicle's ID.

(ii) The source vehicle's velocity vector.

(iii) The source vehicle's idle state.

(iv) A list of the IDs of the source vehicle's idle 1-hop neighbors.

Note that if the list in (iv) is too long, then only a few randomly selected entries are included and the rest are ignored. We choose 50 as the maximum number of selected entries in our implementation, which is sufficient in most cases.

Any receiver of this beacon message can extract the information it carries. Upon receiving a beacon, the receiver calculates the speed difference between itself and the sender as $|\mathbf{v} - \mathbf{v}_0|$, where $\mathbf{v}$ is the received velocity and $\mathbf{v}_0$ is the receiver's velocity. If this speed difference is larger than a pre-assigned value, which is set to $15m/sec$ ($54km/hr$, approximately the speed difference when two vehicles are driving in orthogonal directions on urban streets) in the default design, then the receiver considers its link with the source node to be unstable and ignores this beacon message. Otherwise, the receiver records every vehicle mentioned in the message to its NAI table using the following rules:

- If the vehicle is not in the table, an entry is created for it. *Number of hops* is set to 1 for the source vehicle of this beacon or to 2 for a vehicle in the source vehicle's neighbor list. *Expire time* is updated to $t_{now} + D$, where t_{now} is the current time and D is a configurable duration (where the default is the beacon interval).

- If the vehicle is in the table and the source vehicle of the beacon is recorded as a 2-hop neighbor in the table, then the hop number is changed to 1 and *expire time* is updated accordingly; otherwise, only *expire time* is updated.
- Otherwise, no modification is made to the table.

2.3.2 Job caching

Rather than scheduling and uploading each job immediately upon its arrival, the manager module runs a queue to cache jobs, where each caching duration is called a *caching window*. The main reason for caching is to avoid multiple simultaneous discovery processes and data transmissions, which not only would cause jobs to compete with each other for limited bandwidth but also might make returned messages invalid. This is because the available resources may change as a result of offloading earlier jobs. Moreover, caching allows better assignment decisions and reduces the discovery frequency. When more jobs are collected, we can more effectively find the best processor for each job by considering the job priorities and node utilization statuses.

When the first job arrives in the current window at t_{first}, an end time of $t_{end} = t_{first} + Q/(NAI + 1)$ is set. Here, Q is the estimated time for this requester to process all jobs in its current processing queue, which can also be taken as the "queuing time" if a new job is to be processed locally at the current time. In particular, $Q = 0$ if this requester is idle.

The caching duration ends when both of the following conditions are satisfied: (a) the previous flow has finished data transmission, and (b) either the number of cached jobs is greater than $(NAI + 1)$ or the time exceeds t_{end}. The general idea is as follows: if there are sufficient available vehicles nearby, we want to cache as many jobs as the current *edge* (formed by nearby nodes) can handle, so that the frequency of discovery can be reduced and a better assignment can be found. However, we do not want jobs to wait too long if resources are abundant (i.e., if NAI is high).

To eliminate the overhead incurred by useless caching in sparse areas, caching is skipped if the current NAI is 0. When this occurs, the *flow* is terminated. All jobs are scheduled to be processed locally, in a first-come-first-served manner.

2.3.3 Discovery

When the caching time expires, the current job queue is locked and the discovery phase begins. During the discovery phase, the requester first broadcasts an "edge request message" (EREQ) to discover potential nearby processors. This EREQ contains the requester's ID, its velocity vector, and *briefs* of jobs cached during the last *caching window*.

Each receiver checks the velocity vector contained in the EREQ and calculates the speed difference using the same rule described for beaconing. If the speed difference is too large, this EREQ is ignored. Otherwise, the following procedures are performed. All 1-hop receivers relay the EREQ to nearby vehicles; i.e., they become the *forwarders* for this requester. Moreover, for each job with which the receiver is compatible, a query with the job's *brief* is sent to the *resource manager*. The *resource manager* then asks the corresponding backend to generate a *bid*. The *bid* is an estimate of the time required to finish the job, which is determined based on the necessary computation (specified in the brief provided in the job entry), the processing capacity of the vehicle, and the available computing resources that it is willing to share. We let B_{ij} denote a bid from node i for job j. After all queries are complete, if there are any *bids*, then the receiver sends back an *edge* response message (EREP). The EREP includes the responding vehicle's ID and the aforementioned *bids* for each job. If the responding vehicle is the requester's neighbor, then the EREP is sent back directly. Otherwise, the EREP will be sent to the requester via the *forwarders*. The same route is used in this framework (i.e., a node sends its EREP to the node that relayed the corresponding EREQ to it) since it is reasonable to assume that the network topology will remain the same over such a short duration. To avoid conflicts of requests from two different requesters, each receiver reserves the promised resources for a short duration (the default is 50 ms), during which it pauses in responding to requests. However, relaying is not restricted by this rule.

The requester waits for a short period of time to collect acknowledgments from other vehicles. The wait time is set to the maximum round-trip delay for 2-hop transmission. If this round-trip delay cannot be estimated, a manually configured time (the default is 50 ms) will be set. After the wait time expires, the requester starts job scheduling based on the collected information.

2.3.4 Scheduling

The scheduling problem scenario can be described as follows: After caching and discovery, we have a finite set of jobs and a finite set of nodes. Each job is to be transmitted to one of the nodes to be processed there. First, transmission must be completed before a job can be processed. To reduce the average transmission time, each job's transmission begins only after the previous job's transmission is finished. Because of the limited bandwidth available in a vehicular environment, the transmission time is not negligible. While there is uncertainty when jobs are actually being transmitted, we make an estimation on the transmission time with a moving window average transmission rate over the recent transmissions and the size of individual jobs. When the assigned processor has received all data for a job, it immediately places the job into the processing queue. Queued jobs are processed one by one, in first-come-first-served order. At any given instant of time, there is only one job being executed to allow each job to utilize all available resources. When the processing of a job is finished, the result of the job is transmitted back to the requester. Upon the result being received, the job is considered as finished and the corresponding utility is gained.

This scheduling approach attempts to maximize the total utility by determining *the order of jobs to be transmitted* and *the processor node of each job* (i.e., for each transmitted job, which node is selected to assist in processing). The problem can be treated as a 2-stage hybrid flow-shop problem, which is *NP-hard* in strong sense[53]. The formulation of and proposed solution for the scheduling problem are described in Section 2.4. As scheduling is finished, the requester has the sequence of jobs to be transmitted and the destination node of each job determined. Then, data transmission begins immediately.

After the assignment is complete, a short notification message is broadcast using the same rule as for EREQs. This notification contains the assignment result obtained during scheduling. This is to notify the assigned nodes to stop sending EREPs and wait for the jobs that have been assigned to them to arrive.

2.3.5 Data transmission

The offloading process begins after the scheduling phase finishes. Because the running time of the assignment algorithm is short, it is reasonable to assume that the network structure will not change compared with the network structure during the job discovery phase. Therefore, the data can be transmitted along the same paths used by the EREQs. The order of transmission is determined during the scheduling process, as discussed in Section 2.3.4.

Upon receiving all data for one job, a processor adds that job to the end of its processing queue. Queued jobs are processed one by one. For each job, the *resource management* module brings the corresponding *backend* into its managed VM for execution and then waits until the *backend* finishes the current job. During execution, the *backend* can utilize the promised resources of the managed VM.

After the job process is complete, the results will be sent back to the requester. Since we do not focus on ad hoc routing in this work, we use the traditional routing protocol known as *ad hoc on-demand distance vector (AODV) routing* [55] for this transmission in our implementation. But note that AODV is not the only choice, other unicast protocols, e.g. Predictable Broadcast Reckon (PBR)[56] are also feasible here, among others[57]. Moreover, if the routing path between the two vehicles is available (e.g. established by other vehicular network applications), we can take advantage of this path directly. Also considering that the mobility is also limited by the speed difference constraint during the discovery phase, the routing is even easier to achieve.

2.4 Scheduling algorithm

Scheduling has been an essential problem since the emergence of cloud computing. Algorithms are develop to meet different scenarios' demands[58][59][60][61]. In this section, the ACO-based scheduling algorithm for each node that requires the help of other nodes is explained. AVE is a decentralized framework, and each node individually makes the scheduling decisions for its own jobs based on the proposed scheduling algorithm.

The algorithm considers constraints including the host requirements, and as stated in Section 2.2.3,

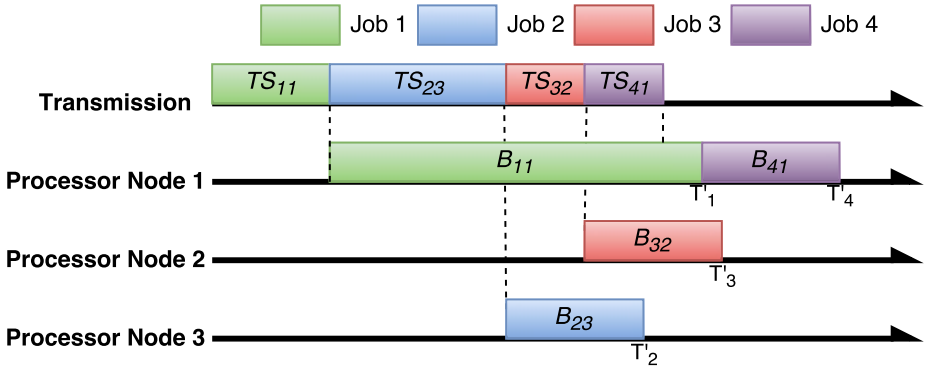


Figure 2.4: An example of the time constraints of the assignment problem

one job should be processed in exactly one location to avoid wasting resources, and time constraints. The time constraints include the following. We assume that there can only be at most one job in transmission at the same time by the DSRC interface or cellular network, even though the DSRC and cellular interfaces are orthogonal. Then, processing can only start after receiving all the job's data. For RSU and vehicle nodes, because they are running a processing queue, the new jobs also have to wait for the preceding jobs to finish.

Fig. 2.4 shows an example. Job 2 is assigned to Node 3; although Node 3 has no other jobs in processing, it has to wait until Job 2 is transmitted to it, which is after the transmission of Job 1. Job 4 is received on Node 1 before T'_1, but it has to wait until Node 1 finishes processing Job 1, which comes before it.

2.4.1 Problem formulation

As described in Section 2.1, the problem can be stated as follows: Two finite sets, J and P, are given, where J is the job set and P is the processor set. For each job $j \in J$, a utility function $\omega_j(t)$ is defined, where t is the completion time of the job (when the result is returned to the requesting application). We do not place any constraints on $\omega_j(t)$ except that it must be a non-increasing function with positive

Table 2.1: Symbols used in formulating the problem

Symbol	Explanation
J	set of jobs
P	set of nodes
S	a found solution
Q_i	queuing time on node i
B_{ij}	required processing time for job j on node i
TR_{ij}	estimated time for job j's result to be returned from node i
TS_{ij}	estimated time for sending job j to node i
$\omega_j(\cdot)$	utility function of job j
$c(i,j)$	compatibility of job j and node i
t_j	completion time of job j
p_j	time when the sending of job j (to its processor) finishes
q_j	queuing time of job j on its processor
o_j	time when job j finishes processing on its processor
$\Omega(S)$	utility gained with solution S

values. Namely, we have the following equation:

$$\omega_j(t_1) \geqslant \omega_j(t_2) > 0, \text{ for all } t_1 < t_2$$

This assumption is based on the fact that earlier completion is better for all jobs. For different jobs (say j_1 and j_2) and the same time t, $\omega_{j_1}(t)$ may not equal $\omega_{j_2}(t)$ because of the priority difference among jobs. The corresponding application provides the ω function for each job, which accounts for the importance of the job and its real-time requirements.

As previously mentioned, not all vehicles will necessarily possess all functions needed to be capable of processing all jobs; $c(i,j)$ is used to denote this situation. For each node i and each job j, $c(i,j) = 1$ if and only if j can be processed on i; otherwise, $c(i,j) = 0$. Note that we assume here that for any job j, there is at least one node that can process it (otherwise, the job is declared to have failed and is discarded before assignment); *i.e.,* $\sum_{i \in P} c(i,j) \geqslant 1$.

The transmission of each job begins only after the previous job has finished. Moreover, for two jobs assigned to the same node, the one that is received later must wait for the completion of the

earlier job before its processing can start. The queuing time before the current scheduling on node i is represented by Q_i. In this framework, because only currently idle nodes will respond, Q_i is 0 for all nodes except the requester itself, which bears the responsibility for all of its own jobs even if it is currently busy. For a node–job pair (i, j), $TS_{ij} \geqslant 0$ represents the length of time required to send the data for job j to node i, $TR_{ij} \geqslant 0$ represents the length of time required for the processor to return the processing result, and $B_{ij} > 0$ represents the length of time required to process the job. B_{ij}, as previously discussed, is a *bid* generated by node i during the discovery phase. TS_{ij} and TR_{ij} are roughly estimated from the current network conditions (including the channel bandwidth and so forth), the number of hops to i and the amount of data to be transmitted for job j. In particular, we have $TS_{ij} = TR_{ij} = 0$ if i itself is the *requester* of j.

A sequence $[(i_1, j_1), (i_2, j_2), \ldots, (i_n, j_n)]$, where $i_k \in P$ $(k = 1, 2, \ldots, n)$ and $\bigcup_{k=1}^{n}\{j_k\} = J$, must be found to

$$\text{maximize} \sum_{j \in J} \omega_j(t_j) \tag{2.1}$$

s.t.

$$j_k = j_h, \text{ i.f.f. } k = h; \tag{C1}$$

$$c(i_k, j_k) = 1, \ \forall k = 1, 2, \ldots, n; \tag{C2}$$

$$t_{j_k} = o_{j_k} + TR_{i_k, j_k}, \ \forall k = 1, 2, \ldots, n; \tag{C3}$$

$$o_{j_k} = max(p_{j_k}, q_{j_k}) + B_{i_k, j_k}, \ \forall k = 1, 2, \ldots, n; \tag{C4}$$

$$p_{j_0} = 0; \tag{C5}$$

$$p_{j_k} = p_{j_{k-1}} + TS_{i_k, j_k}, \ \forall k = 1, 2, \ldots, n; \tag{C6}$$

$$q_{j_k} = max(\{o_{j_h} | i_h = i_k, h < k\} \cup \{Q_{i_k}\}), \quad \forall k = 1, 2, \ldots, n. \tag{C7}$$

Objective function (2.1) represents the total utility gained from all assigned jobs, which is our target

of optimization. Condition (C1) means that any given job cannot be assigned twice in a valid solution. Condition (C2) indicates that each processing node must be compatible with each job assigned to it. Conditions (C3)-(C7) are the aforementioned transmission and processing time constraints. p_{j_k}, q_{j_k}, and o_{j_k} all specify time instants: p_{j_k} is the receiving time, q_{j_k} is the wait time for previous jobs on the assigned node to finish, and o_{j_k} is the time when job j_k finishes processing on the node. We let $\Omega(S)$ denote the value of $\sum_{j\in J}\omega_j(t_j)$ obtained as the outcome of a solution S.

Such a sequence $[(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)]$, denoted by S, is called a *solution* to this problem throughout the remainder of this chapter. If $k = |J|$, this solution is a *complete solution*; otherwise, it is called an *intermediate solution*. Specifically, if $k = 0$, *i.e.,* for the sequence $[\varnothing]$, we call it a *null solution*. In a *solution*, each (i, j) pair represents an assignment of job j to node i. The order of the assignments in a solution is the order in which these jobs are to be sent to the processors.

The problem formulated here is a version of the 2-stage hybrid flow-shop problem[62][63] with more inputs (with stage times that vary with different numbers of assignments) and an objective function that can take any form (a linear function or a reciprocal function, among others). The original 2-stage hybrid flow-shop problem has been proven to be *NP-hard*[53] even with only 2 nodes in the second stage. For the problem considered here, which is more difficult, it is unlikely that a complete optimizing algorithm can be found with a polynomial solution time.

2.4.2 ACO-based approach

Lacking a complete algorithm to solve the problem, we turn to heuristic algorithms, which can yield near-optimal solutions within a reasonable running time. Among many possible heuristic algorithms, e.g. genetics algorithm, simulated annealing, etc., we choose Ant Colony Optimization(ACO) based approach, because: (a) the sequence representation of a *solution* can be transformed to the "path" concept in ACO naturally, and (b) the conditions listed above are easies to be built into the algorithm, as we are going to show afterwards.

Ant Colony Optimization (ACO)

ACO[64] is a methodology inspired by the behavior of ants searching for paths to food. In the natural world, upon finding food, an ant lays down a pheromone trail as it returns to the colony, which then attracts other ants to follow its path. Because pheromones evaporate over time, a longer trail will tend to lose more pheromone as the ant travels back, which makes it less attractive. Conversely, a shorter path will retain more pheromone and attract more ants to follow. These additional attracted ants will also deposit more pheromone, and this positive feedback will ultimately, through iteration, attract all ants to use the shortest path.

When this concept is applied to a mathematical problem, the term "trails" often refers to state transitions. A series of consecutive transitions will lead from the starting state (null solution) to the ending state (complete solution), thereby providing a feasible solution to the problem. Typically, an ACO algorithm has four steps:

(i) *Initialization*: Initialize all trails' pheromone levels.

(ii) *Solution construction*: Set the state to the starting state (from an empty solution). Then, repeat the selection of trails to the next state, with probabilities given by their pheromone levels, until the ending state is reached.

(iii) *Updating*: Update the trails' pheromone levels: (a) reduce all pheromone levels according to the evaporation rate, and (b) increase the pheromone levels of all trails chosen by ants.

(iv) *Termination*: If the termination condition is met, return the best solution. Otherwise, return to Step (ii).

The ACO concept has previously been adopted to solve assignment problems[65] and scheduling problems[66] [67] with satisfactory results. Next, we will show how we solve the job scheduling problem in AVE using ACO.

Definitions of trails and pheromone levels

To utilize the ACO algorithm for our problem, we must first define "trails" and "pheromone" in the AVE framework. Trails should represent components of the final solution, each of which can be optimized locally. $\omega_j(t_j)$ is not a good choice for local optimization because jobs are in competition for resources – when resource allocation is optimized for one job, it will always worsen for other jobs, thereby reducing the chance of improving the global performance. Therefore, we wish to find a better parameter that can represent local optimality while not being bound to specific jobs. Using Condition (C1) in the problem formulation, we can transform the objective function (2.1) as follows:

$$\sum_{j\in J}\omega_j(t_j)=\sum_{i\in P}\sum_{i_k=i}\omega_{j_k}(t_{j_k}) \tag{2.2}$$

since $\sum_{i_k=i}\omega_{j_k}(t_{j_k})$ is the total utility of jobs completed on node i. Now, we let

$$a_i(t)=\omega_{\bar{j}}(t_{\bar{j}})/(o_{\bar{j}}-q_{\bar{j}}), \text{where } x_{i\bar{j}}=1 \text{ and } o_{\bar{j}}>t\geqslant q_{\bar{j}} \tag{2.3}$$

Note that if there is no such $\bar{j}$ that satisfies the conditions, then $a_i(t)=0$. Additionally, there will be at most one $\bar{j}$ that satisfies the conditions. The reason is that for each processor, there will be at most one job (say j') being processed at one time, which starts at $q_{j'}$ and ends at $o_{j'}$. The real-world meaning of $a_i(t)$ can be understood to be the "amortized utility". The idea is to spread the utility gained from finishing a job throughout the processor time it requires, which clearly includes the processing time and also includes the wait time if its data have not arrived before the completion of the previous job.

We can see that $\int_{q_{\bar{j}}}^{o_{\bar{j}}}a_i(t)dt=\omega_{\bar{j}}(t_{\bar{j}})$. Thus, it is easy to obtain

$$\sum_{i_k=i}\omega_{j_k}(t_{j_k})=\int_0^{+\infty}a_i(t)dt \tag{2.4}$$

Note that although the integration here is performed on $[0,+\infty)$, this does not mean that we are scheduling up through a time of $+\infty$. This is because the jobs that we are considering right now, *i.e.*,

the set J, are finite. With (2.4) and (2.2), the objective function (2.1) can be rewritten as

$$\text{maximize} \sum_{i \in P} \int_0^{+\infty} a_i(t)dt \tag{2.5}$$

Now, the goal of optimization is transformed into the maximization of the value of $a_i(t)$. $a_i(t)$ is a better parameter for measuring "trails" in ACO because it not only can be locally optimized but also contributes to global optimization. Thus, we define a trail as follows:

Definition 1. A *trail* $z(a, b)$ is a state transition that augments a null or intermediate solution $S = [(i_1, j_1),(i_2, j_2),\ldots,(i_k, j_k)]$ to another solution $S' = [(i_1, j_1),(i_2, j_2),\ldots,(i_k, j_k),(i_{k+1}, j_{k+1})]$, where $i_{k+1} = a$ and $j_{k+1} = b$.

The trail concept is illustrated in Fig. 2.5. A solution is constructed is as follows: The initial *null* solution is set to $S^* = [\varnothing]$. A complete solution $S = [(i_1, j_1), (i_2, j_2), \ldots, (i_n, j_n)]$, where $n = |J|$, is found by performing n consecutive transitions.

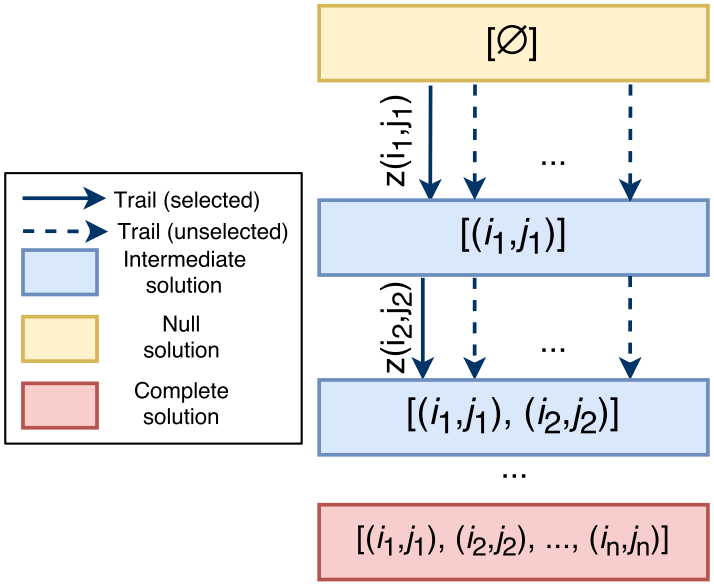


Figure 2.5: Example of trails and solution construction.

The desirability of selecting transition $z(i, j)$ in state S is given by $a_i(q_j)$. Here, the starting

point q_j is chosen to represent the corresponding value in $[q_j, o_j)$. The current level of deposited "pheromone" for (i, t) (where $i \in P$ and $t \in [0, +\infty)$) is denoted by $\tau_i(t)$. This value is used as follows: if the desirability a of a trail is lower than the corresponding τ, it is multiplied by a/τ; otherwise, the desirability a remains the same. The rationale behind this calculation is that we attempt to reduce the possibility of selecting a lower a such that the search can be guided toward solutions with higher objective values. Although forms other than a/τ might be feasible, we choose this factor because it is simple and is not affected by the scale of the utility values (e.g., if the value of ω were to be enlarged by 10 times for all jobs, a form such as $a - \tau$ would also yield probability differences that were 10 times higher).

The "pheromone" level τ starts at 0. For each successfully found solution, if it improves upon the best known result, then "pheromone" with an initial amount equal to $a_i(t)$ will be deposited at the end of the current iteration. Additionally, in each iteration, the deposited "pheromone" will evaporate at a constant evaporation rate, thereby reducing the value of τ.

Therefore, we can summarize the overall procedure as follows: we first *initialize the parameters* and then repeat the process of *constructing solutions* and *updating the parameters* until the termination condition is met, i.e. when the best solution is returned. Further details of each step are described in the following section.

2.4.3 Detailed algorithm

Initialization of parameters

The first step of the ACO algorithm is to set the initial parameters. In this step, the best solution, denoted by S^*, is set to null ($[\varnothing]$), and all levels of deposited "pheromone", which are represented by $\tau_i(t)$, are set to 0 for all $i \in P$ and $t \in [0, +\infty)$.

Solution construction

Then, we construct a complete solution via local optimization using Algorithm 1.

The algorithm iterates through all compatible combinations (i, j) of nodes and jobs to obtain the

Algorithm 1: Solution Construction

Data: "Pheromone" level τ_i
Result: Complete solution S

```
1  Initialize current solution S = [∅];
2  Initialize set of unassigned jobs J' = J;
3  while J' ≠ ∅ do
4      for j ∈ J' do
5          for i ∈ P do
6              if c(i, j) = 1 then
7                  Calculate the value of a_i(q_j) when transition z(i, j) is applied to S;
8                  if a_i(q_j) < τ_i(q_j) then
9                      η_ij = (a_i(q_j))^2 / τ_i(q_j);
10                 else
11                     η_ij = a_i(q_j);
12                 end
13             else
14                 η_ij = 0;
15             end
16         end
17     end
18     Select a pair (i, j) by sampling from the probability distribution
         ρ_ij = η_ij / (Σ_{i'∈P} Σ_{j'∈J'} η_{i'j'});
19     Augment S by applying z(i, j) to it;
20     Remove j from J';
21 end
```

values of the desirability η_{ij}. As discussed in the definitions, we use the amortized utility $a_i(t)$ for this purpose. The calculation of $a_i(q_j)$ is inexpensive. Essentially, the implementation can choose to keep track of the values of q for the last job assigned during this iteration and q for the last job assigned to node i during this iteration. Then, using Equation (2.3), we obtain $a_i(q_j)$ with $O(1)$ time complexity. In Line 9 $\sim$ 14, η is the final desirability, or the weight, for selecting a transition. We can see that in the first iteration, η will be directly set equal to the corresponding a, which provides this iteration with a good starting point for faster convergence. Then, we sample from the probability ρ_{ij} to select j as the next job to send and i as the processor to which to assign it. In the final step, j is removed from J' such that Condition (C1) is satisfied.

The conditions listed in problem formulation are satisfied as following:

(i) Condition (C1) is satisfied in Line 20, where each job assigned gets removed from the set and

avoids duplicated assignment.

(ii) Condition (C2) is satisfied in Line 6. All selected pairs of node i and job j are guaranteed to have compatibility.

(iii) Conditions (C3) $\sim$ (C6) are satisfied in the calculation of q_j (Line 7) and augmenting operation (Line 19). In detail, when an augmenting operation is to be done, we calculate $p_{j'}$ (j' is the last job in current solution before augmentation, i.e., j_{k-1} in the formulation if we take current j as j_k) and q_j with current incomplete solution S; then, p_j and o_j are calculated with (C6) and (C4); note that Line 1 where we start a solution from $[\varnothing]$ implies (C5) is satisfied.

Parameter update

A complete solution S is found in the previous step. Then, the parameters are updated to guide the next iteration to a possibly better solution. The updating algorithm is presented as Algorithm 2.

Algorithm 2: Parameter Update

Data: "Pheromone" level τ_i, best solution S^*, last solution S, evaporation rate λ
Result: Updated τ_i and S^*

```
for i ∈ P do
    Update τ_i(t) to (1 − λ)τ_i(t) for all t ∈ [0, +∞);
end
if Ω(S) ⩾ Ω(S*) then
    Set the new best solution S* to S;
    for i ∈ P do
        Calculate a_i(t) using solution S for t ∈ [0, +∞). For any t such that a_i(t) > τ_i(t), set
         τ_i(t) to a_i(t);
    end
end
```

The first loop of this algorithm simulates the evaporation process. All thresholds lose some of their strength, thereby increasing the probability that alternative solutions will be sought. The parameter affecting this process is the *evaporation rate*, denoted by λ. It is chosen to be 0.4 by default. As tested on our arbitrarily generated cases, the algorithm achieves a higher utility on average when this value is used. The second loop will be called when the last constructed solution is the new best result. This

new best solution is used to update the thresholds to be at least equal to the amortized utilities. With this update, we can guide the subsequent iteration toward a better solution.

In updating the parameters, the stored best solution is not modified. Therefore the conditions (C1) $\sim$ (C7) are still satisfied.

Termination

After an iteration is completed, the termination condition is evaluated. A typical strategy is to end the optimization after a sufficient number of iterations or when the improvement of $\Omega(S^*)$ over the last several iterations is less than a desired value. If the termination condition is not met, we return to Step 2 (Algorithm 1) with the updated τ_i and S^* value for an additional iteration.

2.4.4 Algorithm analysis

Suppose that the number of jobs is n and that the number of nodes is m. In each iteration, one solution is constructed. Each solution consists of n assignments (trails). When selecting a trail, a maximum of $n \times m$ pairs are examined, each with $O(1)$ time complexity. Therefore, the time complexity of one iteration is $O(n^2m)$. Additionally, the updating of the parameters can be performed in $O(n)$ time. Thus, for t iterations, the total time complexity of the proposed algorithm is $O(tn^2m)$.

However, in practice, the convergence rate is a greater concern than the per-iteration complexity. For a preliminary evaluation of the convergence rate, we (a) arbitrarily generated jobs and nodes with randomly given parameters, (b) solved the scheduling problem using both the proposed ACO algorithm and a brute-force method, and then (c) calculated the ratio of the resulting total utility to the optimal one. Because a brute-force search has a time complexity of $O(n!m^n)$ (n permutations of jobs and m^n assignment alternatives), the numbers of generated nodes and jobs could not be too large. We selected 5 jobs and 5 nodes for this evaluation. We generated 50 cases with random parameters, and each case was run 50 times to obtain the average outcome. We also employed a random assigning algorithm, which forms a legal solution randomly in each iteration, as a competing algorithm.

The results are shown in Fig. 2.6. This figure shows that the proposed algorithm achieves, on

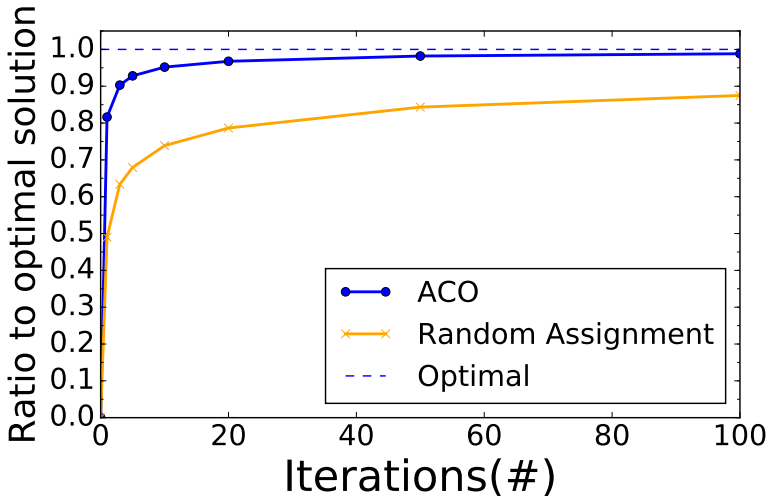


Figure 2.6: Convergence rate in preliminary algorithm evaluation.

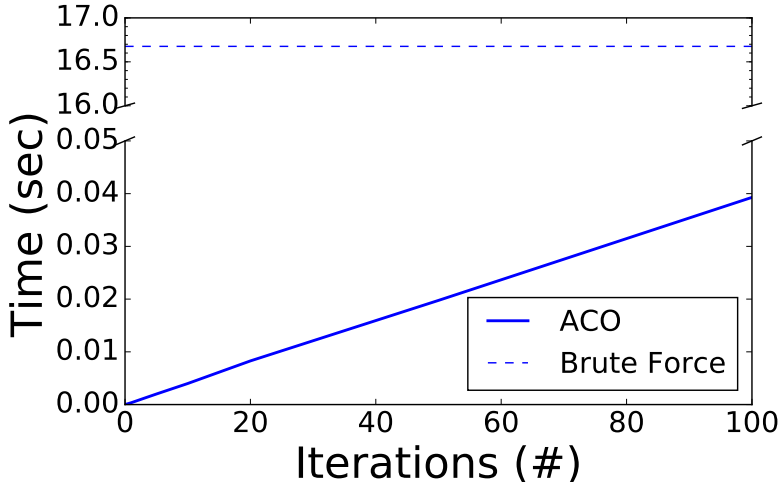


Figure 2.7: Running time in preliminary algorithm evaluation.

average, 95% of the utility of the optimal solution within fewer than 10 iterations, which demonstrates the fast coverage and good performance of the proposed scheduling algorithm. Moreover, from the results of random assignment, we can observe that 100 random iterations in the search space can achieve only 87% optimality. In addition, as tested on a desktop computer with a 3.0 GHz CPU and 8 GB of RAM, the algorithm requires approximately 40 ms to run 100 iterations in Python implementation,

with the same input as above, whereas a brute-force search requires more than 15 seconds. Note that this is only a preliminary test; we will show how the algorithm works in practice in Section 2.5.

2.5 Evaluation

In this section, we describe how we set up the simulations to evaluate the performance of the proposed framework in various scenarios, and we report the evaluation results with respect to competing schemes.

2.5.1 Simulation setup

The simulation setup is introduced in terms of the traffic, network, application settings and competing schemes considered for comparison. An overview of the tools used in simulation is shown in Fig. 2.8. There are four main notable parts: the network simulation environment *Omnet++*[68], the road traffic simulator *Simulation of Urban MObility* (SUMO)[69], the framework *Veins*[70] to connect the previous two, and the *INET* package for some network layers implementation. The details of them are stated in the following sections.

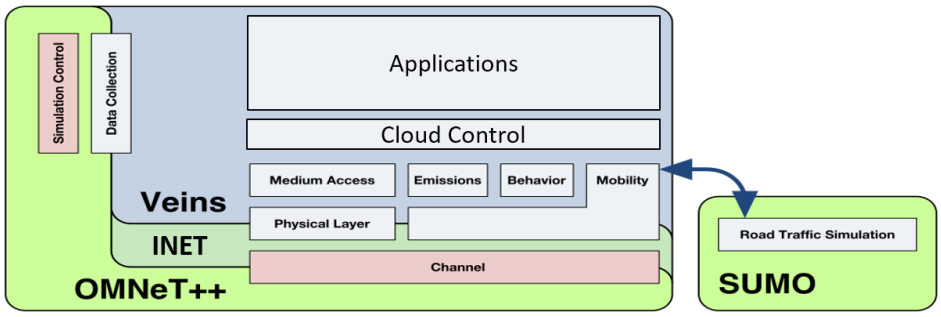


Figure 2.8: The stack of software used in the simulation.

Simulation tools and traffic simulation

To better emulate real traffic, we used *Simulation of Urban MObility* (SUMO)[69] as the traffic simulator. The *Luxembourg SUMO Traffic scenario* (LuST)[71] was used to generate the maps and the traffic. The geographic map is a map of Luxembourg City, Luxembourg. The scenario covers an area

of 155.95 km^2 and more than 24 hours of traffic. We chose two representative regions of this large map. The geographical areas of the two selected regions are shown in Fig. 2.9 by the blue lines and the green block. As we can see from this map, these two regions represents *highways* and *urban roads*, respectively. On highways, vehicles move faster, and the inter-vehicle distance is longer, which results in a lower density. However, there are also considerably fewer obstacles (e.g., buildings) in a highway scenario. We tested the performance of the proposed framework in these two representative scenarios. We chose a time period of 15 minutes, from 6:00 *am* to 6:15 *am*, during which the traffic demand is neither too high nor too low, as shown by the blue bar in Fig. 2.10. Please note that the proposal still works in higher or lower traffic demand. But some modification to the parameters (e.g. beacon interval) might be needed for optimizing performance.

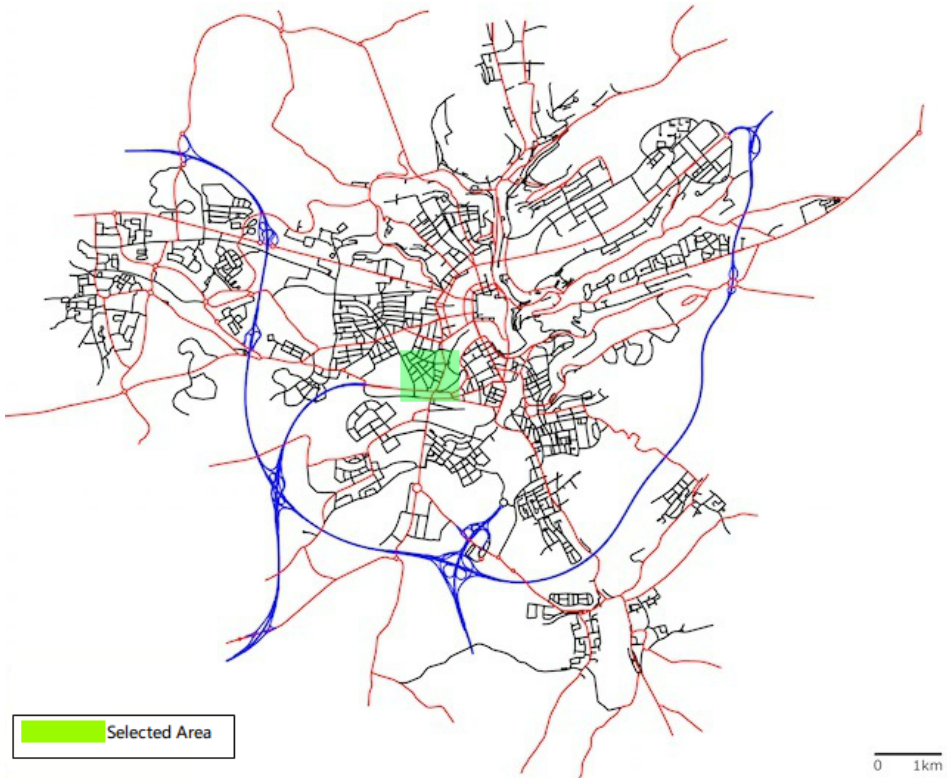


Figure 2.9: Map of Luxembourg used as the traffic map in the evaluation. The blue lines delineate the highway region, and the area marked with a green block in the center was used for the urban scenario. *Figure source: LuST*[71] (modified).

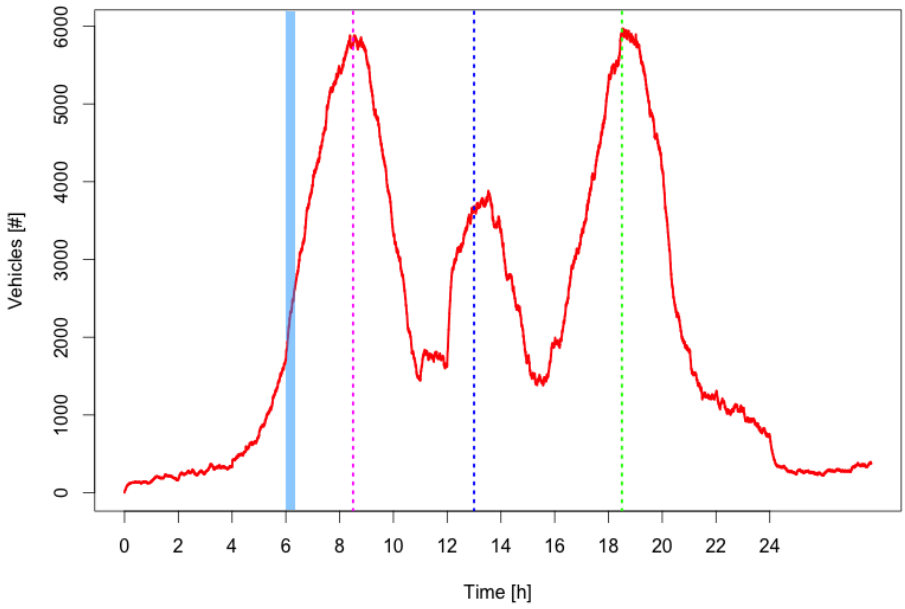


Figure 2.10: Traffic demand in the *LuST* scenario. The time window selected for the simulation (6:00 *am* to 6:15 *am*) is marked with a blue bar. *Figure source: LuST*[71] (modified).

Network

The well-known IEEE 802.11p DSRC standard was used here for the connections between vehicles. We simulated IEEE 802.11p DSRC using the *Omnet++*[68] simulator, together with *Veins*[70], a network framework for 802.11p. The carrier frequency was set to the 5.9 GHz band. The default transmission power of the vehicles was set to 20 mW, for which the maximal transmission radius in our simulation is approximately 500 meters. In this simulation, one random service channel was selected, and all service channels had an equal probability of being selected.

We left the MAC layer and physical layer to be simulated with default modules provided by *Veins*. The MAC module implements priority queues, CSMA/CA and channel switching, according to the IEEE 802.11p standard. On the physical layer, for simulation performance, a simple path loss signal model[72] and shadowing model[73] were used. The shadowing model considers each wall of buildings and applies a constant attenuation to the signal, and each meter propagated within a building also applies a constant attenuation. Then a simple path loss model applies an attenuation according to

wave length and distance between sender and receiver. Therefore the receiver's signal strength is calculated with:

$$P_r = P_t + 10 \lg \left(\frac{G_t G_r \lambda^2}{16\pi^2 d^\alpha} \right) - \beta n - \gamma d_m$$

Here, P_r is receiver's signal power, measured in dBm, and P_t is the sender's transmission power, in the same unit. G_t and G_r are sender's and receiver's antenna gains, respectively. λ is wavelength of the signal, and d is the distance between sender and receiver. n is number of obstacles' borders that the signal penetrates, and d_m is the total distance within those obstacles. α, β and γ are all constant parameters. According to aforementioned researches, they are set as $\alpha = 2.2$, $\beta = 9dB$ and $\gamma = 0.4dB/m$.

After that, bit error rate is calculated with SINR[74], the result will determine if the frame is received. Frame collisions when network is congested are also considered in this step.

However, due to restrictions in computation complexity, small scale fading effects like multipath problem and Doppler effect are not considered in the simulation. We will discuss their impact in Section 4.1.

Application settings

To provide universality for applications, the following scenario was chosen for consideration: Each vehicle is moving toward its destination. During driving, the driver or the passengers periodically activate applications and generate jobs; e.g., the passengers activate an AR application to look for interesting nearby objects and shut down this application once the request has been satisfied. Such applications are selected based on personal preference; hence, they are used independently by different users. When an application is activated, requests are generated. Each request is partitioned into several context-free jobs [24] (so that it can be offloaded) to be run in the background, and these jobs are submitted to the framework. To reflect their impacts on the user experience, the application assigns weight factors to the jobs. Then, it issues a request to the framework to process the jobs.

While this framework also works for other models; we select one as an example. The job generation model is as follows. Each application has two states: *idle* and *busy*. Jobs are generated only in the busy

Table 2.2: Application parameter settings in the simulation

Parameters	Default values (if not explicitly set)
I_S	50 sec
R	$R \sim unif(1, 6)$
n	3
I_R	3 sec
u_j	$Pr(u_j = 1) = 0.6$, $Pr(u_j = 2) = 0.3$,
	$Pr(u_j = 3) = 0.1$
s_j	100 KB
s'_j	100 KB
l_j	Exponential distribution with a mean of 4 sec
r_i	$r_i \sim unif(1, 2)$
$\omega(t)$	$l_j/(t + l_j)$

state. A node switches from the *idle* state to the *busy* state following a *Poisson process* with a mean arrival rate of $1/I_S$, where I_S is the expected value of the interval. In the busy state, R requests are made, each containing n jobs. Request arrival in the busy state also follows a *Poisson process*, with an arrival rate of $1/I_R$. After all requests have been made, the application transitions into the *idle* state. Each job j has attributes s_j and s'_j, which denote the sizes of the data required to be transmitted for the job to be offloaded to other vehicles and for the results to be downloaded from the other vehicles, respectively. l_j is the workload. Each vehicle has a *processing rate* r_i; thus, the processing time B_{ij} can easily be estimated as l_j/r_i. The utility function of job j is set to $\omega_j(t) = u_j\omega(t)$, where u_j is its weight factor and $\omega(t)$ is a unified function, which is $l_j/(t+l_j)$ by default. The t used in this function is the time elapsed after the generation of the job. The compatibility $c(i, j)$ is randomly set as follows: if i is not j's requester, then $Pr(c(i, j) = 1) = 0.75$; otherwise, $c(i, j) = 1$. This means that all requesters are capable of processing their own jobs locally.

The parameters of this simulation are listed in Table 2.2.

These default values are set with an example image recognition application in mind, like the one in [75]. The imaginary application, when activated, takes photo with vehicle's front camera, splits it into several regions and conduct recognition algorithms on each of them. Each region has a data size of 100 KB and an average processing time around 3 seconds (mean of l_j divided by expect value of r_i). Recognition tasks has different priorities, e.g. notable marks and spots has higher probability than

background sceneries. And those tasks require a large database of features, which cannot be completely fit into one vehicle, and limits the compatibility of jobs. Note that these values were arbitrarily set because they vary among different applications. For essential parameters such as the workload and utility function, various values were investigated. The influence of other parameters that directly or indirectly affect the workload can be deducted from the results of these investigations.

Competing schemes

Two competing schemes were selected to aid in evaluating the performance of the proposed scheme. The first one is referred to as the *local* scheme, in which only local processing is adopted. In *local*, there is no framework that allows vehicles to help each other. All jobs must be processed on the nodes that generate them. The other scheme we chose uses the proposed AVE framework but not the ACO-based scheduling algorithm. Instead, this scheme uses a simpler scheduling algorithm, which greedily assigns the jobs with the highest utility to the fastest nodes (the nodes with the shortest $TS_{ij} + B_{ij} + TR_{ij}$). This scheme is named *AVE+Naive* and has a lower computational complexity. The proposed scheme is labeled *AVE+ACO*. For the ACO algorithm used here, the termination condition is defined as "the number of iterations reaches 10".

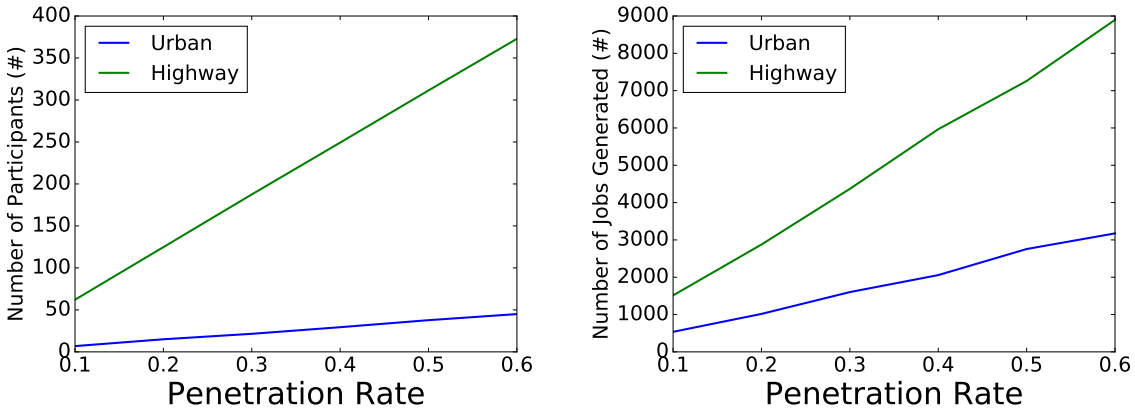
The results shown in each figure are the average results from 10 runs. In each run, all schemes are applied to the same jobs generated at the same time points. We also show 95% confidence intervals in the related figures.

2.5.2 Simulation results

The penetration rate is often used to describe the ratio of active wireless devices within an area [76]. Here, we use this term to refer to the ratio of vehicles participating in the framework; e.g., a penetration rate of 0.1 means that 10% of the vehicles (randomly chosen) on the road are participating. In these two scenarios, various penetration rates are considered to test the framework's performance.

To provide a general picture of the evaluation, in Fig. 2.11(a) and Fig. 2.11(b) we show the average number of vehicles participating in the framework and the total number of jobs generated, respectively.

It can be seen that the number of jobs increases proportionally with the penetration rate, because more participants also mean more requesters.

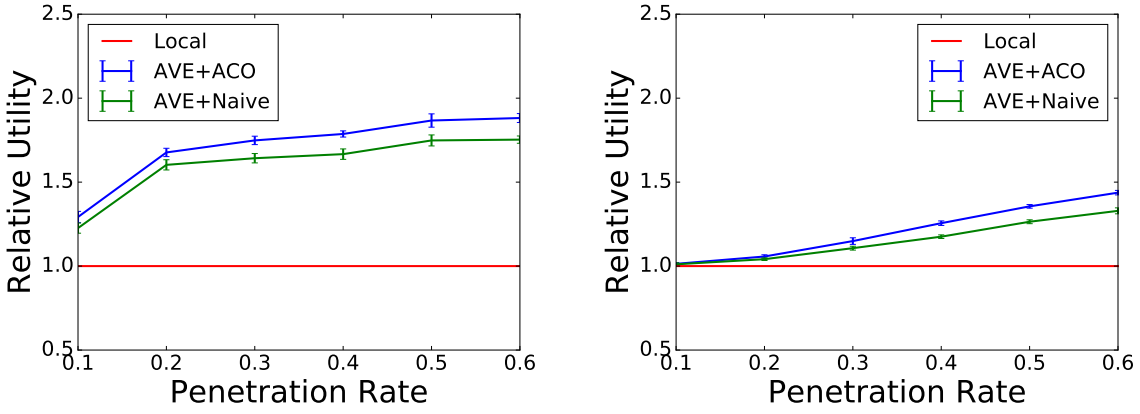


(a) Number of participating vehicles

(b) Number of generated jobs

Figure 2.11: Parameters of the evaluation.

First, we evaluated the total utility gained in these two scenarios under various penetration rates, and the results are shown in Fig. 2.12(a) and Fig. 2.12(b). Given that a higher penetration rate (more participants) also means that more jobs are generated, the results have been normalized to avoid interference from the total number of jobs to facilitate interpretation. This normalization was performed by dividing the results by the corresponding results for the *local* scheme. Thus, in the following discussion of these two figures, the base *local* scheme is associated with values of 1.0. The resulting ratio is called the "relative utility" in the following discussion and is represented on the y axis in these figures. Fig. 2.12(a) shows the results for the urban scenario. As shown in this figure, when more participants are involved, AVE performs better. Additionally, with more participants, greater flexibility is gained in scheduling; hence, the ACO algorithm achieves a higher utility gain over the competing schemes. Fig. 2.12(b) shows the simulation results for the highway scenario. The trend is similar to that observed in the urban scenario. However, because of the higher mobility and longer inter-vehicle distances, the neighboring vehicles are fewer in number and the connections between these vehicles are more unstable in the time domain; hence, the increment is smaller.



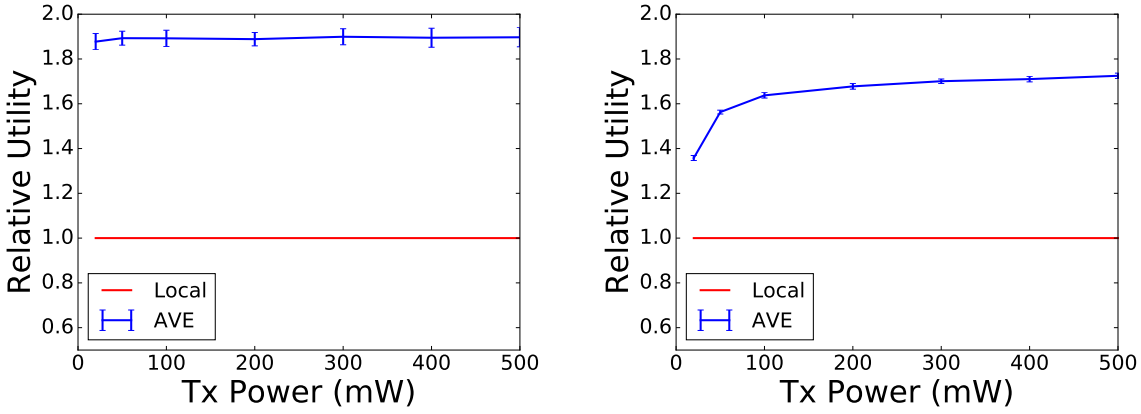
(a) Urban

(b) Highway

Figure 2.12: Relative utility, normalized by dividing the utility gained by that in the purely local scheme.

We also investigated the effect of the transmission power in both scenarios. The results are shown in Fig. 2.13(a) and Fig. 2.13(b). As can be seen, a higher transmission power is actually helpful in the highway scenario because the vehicles can reach farther and find more resources. By contrast, in the urban scenario, the main restriction on communications is the blockage from buildings; therefore, a higher transmission power does not help in this case.

Then, we examined how the workload length (l_j) affects performance; the results are shown in Fig. 2.14(a) and Fig. 2.14(b). We can observe that for extremely short jobs, the advantages of offloading and scheduling cannot compensate for the overhead incurred for discovery and transmission. At the 0.2 workload setting, in which a job requires 0.1 to 0.2 seconds for a vehicle to complete, the average outcome with the AVE framework is even slightly worse than that achieved without it. However, for longer jobs, AVE still provides a significant improvement, as observed from the figures. This is reasonable since for micro-jobs, the overhead for job discovery, data transmission, and so forth is relatively larger compared with their computational requirements than it is for macro-jobs. Here, the term micro-jobs refer to jobs with short workloads, such as function codes that run for hundreds of milliseconds, whereas macro-jobs are those with long workloads, such as data processing tasks that



(a) Urban (b) Highway

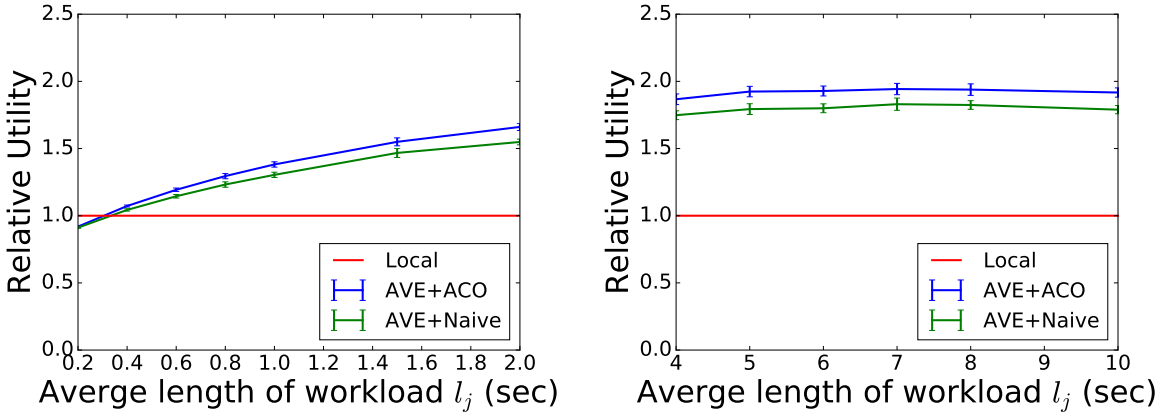
Figure 2.13: Relative utility under different transmission power settings. The penetration rate is set to 0.5.

require seconds to run.

To demonstrate how the results are influenced by the utility function, we used a linear function (Fig. 2.15(a), with a minimum utility value of 0.01 to satisfy $\omega_j(t) > 0$) and an exponential function (Fig. 2.15(b)) as the utility function in the urban scenario. The outcome values are slightly different, but the *AVE+ACO* scheme still outperforms the other two.

For the evaluations discussed below, we set the jobs to have the same utility function (equal to u_j for all j) to allow them to be scheduled without bias.

Fig. 2.16(a) and Fig. 2.16(b) show the proportion of jobs offloaded to other nodes for completion. Because the job generation is the same in all cases, these results can be used to evaluate the extent to which the cloud resources are utilized. As shown in the figure, in the urban scenario with AVE and the ACO algorithm, the ratio rapidly increases with an increasing penetration rate. The ratio reaches its maximum value (0.71) when the penetration rate is 0.6; when the penetration rate is 0.6, 71% jobs are successfully offloaded, whereas the percentage is 63% for *AVE+Naive*. In the highway scenario, these values are smaller, but the ACO algorithm still achieves a higher offloading ratio than *AVE+Naive* does.



(a) Short workload

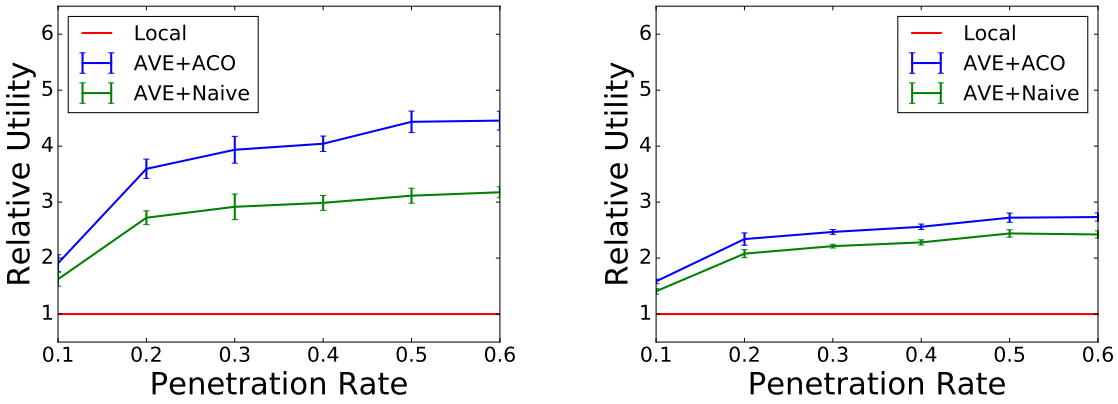
(b) Long workload

Figure 2.14: Relative utility under different settings in the urban scenario.

To compare the improvement in terms of reducing the response times for applications, we define the *Application Delay Ratio* (ADR) [33] as

$$ADR = \frac{\text{total time taken to finish a job}}{\text{minimal time required to finish the job locally}}$$

Here, the *minimal time required to finish the job locally* is the time required for the job to be processed by its own requester with all resources available and no queuing delay. Therefore, ADR can be used as a metric to evaluate the extent to which the framework reduces the response time, without any influence from the different lengths of different jobs. For a cooperative system such as AVE, if all nodes have identical processing speeds and no job has any transmission overhead or queuing delay, then the expectation value of ADR is 1. In Fig. 2.17(a), we show the average ADR values of the three schemes for different penetration rates. As shown, both AVE-based schemes quickly converge to a value close to 1 (not exactly 1 because of transmission overhead). However, in the highway scenario, as shown in Fig. 2.17(b), such convergence is not reached because of the lower vehicle density. Moreover, it can be observed that the difference between *AVE+Naive* and *AVE+ACO* is not always the same. The reason is that when the density of active vehicles (density of vehicles multiplied by the penetration



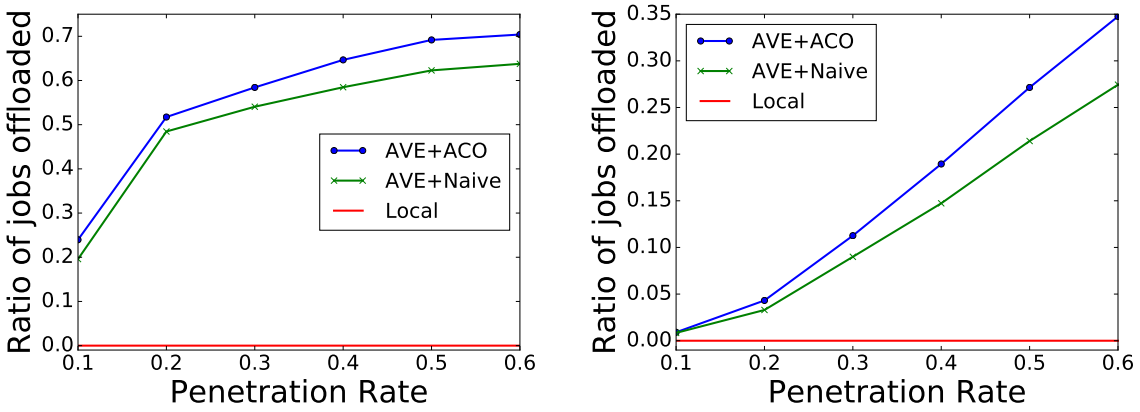
(a) $\omega_j(t) = max(u_j(1 - t/l_j), 0.01)$

(b) $\omega_j(t) = u_j e^{-t/l_j}$

Figure 2.15: Relative utility for alternative utility functions in the urban scenario.

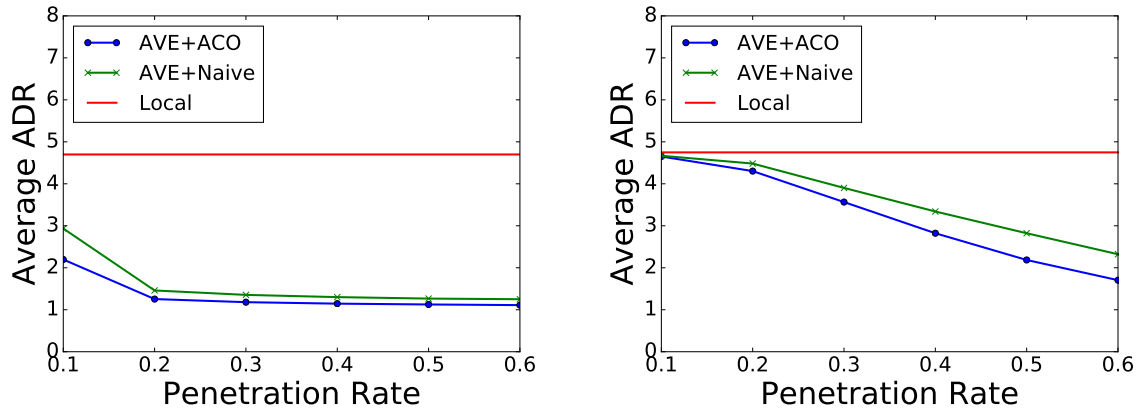
rate) is extremely low, such as in the leftmost region of Fig. 2.17(b), the performance of both algorithms is limited by the lack of neighbors providing assistance, whereas when the density is very high, each processor will be processing only one or two jobs; therefore, even a simple greedy algorithm can achieve an ADR of near 1.0.

We also studied how closely the proposed algorithm's result approaches the theoretical optimal solution. The optimal solution was calculated using the brute-force algorithm by searching all possible options. Because of the brute-force algorithm's high time complexity (of $O(n!m^n)$), we were only able to obtain the theoretical optimal assignments at smaller scales. Therefore, problems with $n!m^n > 500000$ (where n is the number of jobs and m is the number of nodes) were ignored. We brute-force searched for the optimal solutions $S_{optimal}$ for all remaining problems and then compared them with the solutions S^* found in the simulations. The distribution of $\Omega(S^*)/\Omega(S_{optimal})$ is shown in Fig. 2.18. As shown, in over 60% of the cases, the proposed algorithm achieves the optimal solution within 10 iterations, whereas the *naive algorithm* is less than 40% optimal. The computation time of the aforementioned algorithms is shown in Fig. 2.19. The proposed algorithm requires relatively high computation time compared with the greedy algorithm. However, in most cases, the computation time for the ACO algorithm is less than 1 msec, which is much lower than either the transmission latency (5 to 1000



(a) Urban (b) Highway

Figure 2.16: Proportion of offloaded jobs.



(a) Urban (b) Highway

Figure 2.17: ADR under different penetration rates.

msec) or the processing time (200 to 10000 msec) in this case.

As aforementioned, these results are based on simulations without small scale fading. When it is considered, we may have lower transmission rate than simulated. Therefore transmission time of jobs will be longer. This will drive ACO algorithm to prefer locally process the jobs. So that we will see lower ratio of jobs offloaded in Fig. 2.16. And overall lower relative values (normalized by *local* scheme)

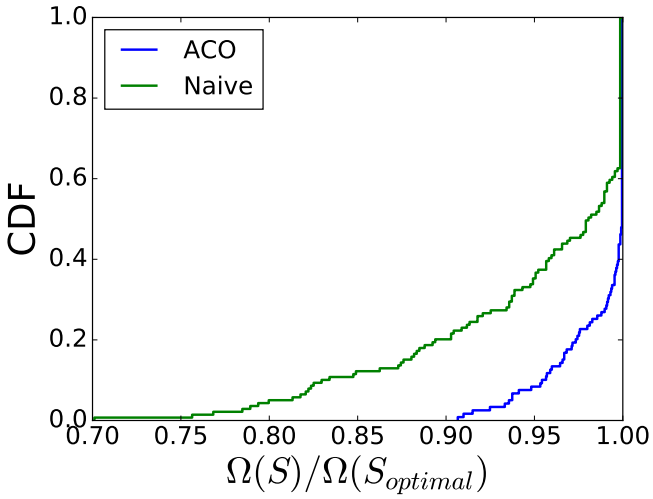


Figure 2.18: CDF of $\Omega(S^*)/\Omega(S_{optimal})$ in the urban scenario with a 0.5 penetration rate.

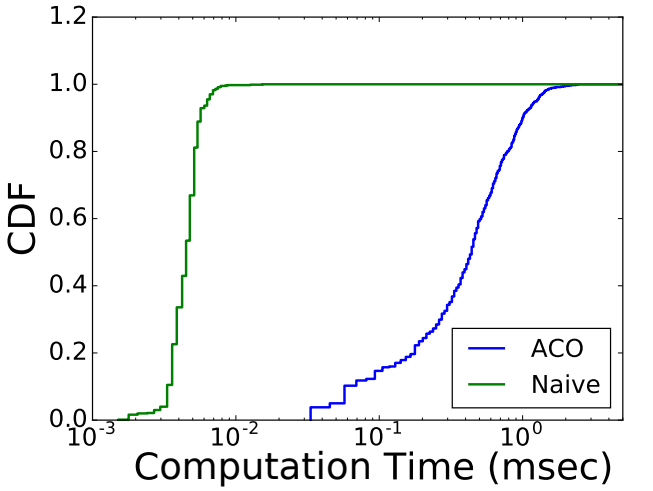


Figure 2.19: Computation times of the algorithms.

in Fig. 2.12∼2.15, as well as higher ADR in Fig. 2.17.

2.6 Summary

In this chapter, a framework named *Autonomous Vehicular Edge* (AVE) is proposed with the aim of increasing the computational capability of vehicles. This framework specifies the workflow of vehicles and how idle computing resources are managed without using deployed infrastructures or centralized control. Arrived jobs are cached according to the job caching scheme, and then an ACO-based scheduling algorithm is proposed to assign the jobs efficiently. A simulation environment is constructed based on state-of-the-art simulation tools to evaluate the performance of the proposed scheme compared to competing schemes. The simulation results show that the proposed framework outperforms the existing schemes in terms of total utility, job delay and workload balancing.

Chapter 3

HVC: Hybrid Vehicular Edge/Cloud Computing Framework

3.1 Problems and goals

In the previous chapter, we consider a vehicular cloud computing framework relying solely on the local resources on vehicles and nearby vehicles, without any consideration about roadside units and centralized cloud access as they are not available.

As a further step, we consider the scenario when such infrastructures are available, and the problem in scheduling among them. This time we also take the cellular network and traditional cloud server into account, and try to address the real-time requirement problems reside in AVE framework. The scenario is detailed below.

3.1.1 Application scenarios

The application scenario is set in the well developed urban environment. Hence in addition to direct DSRC communications between vehicles, cellular network can also be utilized to get access to remote cloud services. But then we have another problem that unlike ad-hoc DSRC network, cellular network

Table 3.1: Application Scenarios for AVE and HVC

	Applicable environment	Application specification	Fitting application
AVE	Urban or rural, No RSU and cellular access, Low penetration	Flexible job finishing time, Delay tolerant	Speech recognition, Route recommendation
HVC	Urban, With cellular access or RSU, Relative higher penetration	Fixed deadlines, Real time	Augmented reality, Games

is generally not free to use. Most providers charge users by the volume of data transmitted through cellular network, and in some cases, the cost of data transmission is higher than the user satisfaction gained from finishing jobs. Also, apart from vehicles, well-developed areas may attract companies to deploy powerful roadside units (RSUs) , which also provide computational services. This also brings more entities to the organization and scheduling. Apart from those, more available resources also allow applications with higher real time requirement to be run. In such a *hybrid* environment, the AVE framework shows its disadvantage for just using nearby vehicles as processors, taking extra time to discovery nodes and allowing a flexible but possibly too late job finishing time.

To solve the aforementioned problems, we propose a framework named *hybrid vehicular edge/cloud framework* (HVC) for cloud computing on the road.

To better illustrate the difference between HVC and AVE, we list some application scenarios in Table 3.1. Note that those are not strict rules, but recommendations. For example, HVC can also work in rural area, without cellular connections, but it may perform worse than AVE in such case, because it is not good at utilize sparse distributed resources; AVE can also handle jobs with fixed deadline by using a step function as utility function, but its scheduling takes more computation time. Speech recognition, when using in casual cases, is flexible in finishing time. But it also requires real time processing when controlling vehicle with voice.

3.1.2 Goals

Similar to previous work, the goal of this framework is to support generalized client-server applications. But this time as we are considering the cellular data usage and RSUs, some differences are presented.

Firstly we need to finish jobs as many as possible, in a reasonable time (set by applications), while keeping cellular network cost low. Secondary with RSUs, which has higher relative speed to vehicles, the link duration is more critical problem in offloading.

Also considering the overhead of job caching and discovery in AVE framework, which is necessary when resources are rare, actually counteracts the performance when we have much more computational resource available. This time we also want a workflow that is more real-time oriented.

With the objective of processing more jobs and reducing the cellular communication cost, an online algorithm is proposed to solve the unique scheduling problem in this framework with various job requirements (e.g., real time and host requirements). The mobility in the vehicular environment is also taken into account.

3.2 System overview

In this section, we introduce each component of the HVC framework, including the roles of the nodes, the software architecture, and the job specifications.

3.2.1 System architecture

Like in AVE, we refer to each participant in this framework as a *node*, including our newly introduced entity, RSUs and remote cloud. However, it should be noted that unlike vehicles can be both *requester* and *processor* role in the previous work, RSUs do not run applications themselves so they can only be *processor*. The same goes to remote cloud in this system.

The software architecture on vehicles is similar to AVE. But with the introduction of cellular offloading, job caching is no longer efficient due to its overhead, hence job queue is removed and all jobs can be scheduled in real-time as they arrive.

Also, in this proposal, positioning devices, such as GPS and map information, are needed to obtain the vehicle's velocity and position. This assumption is reasonable due to the popularity of such devices in vehicles. The location information of the RSU is fixed and is known by itself prior to deploying.

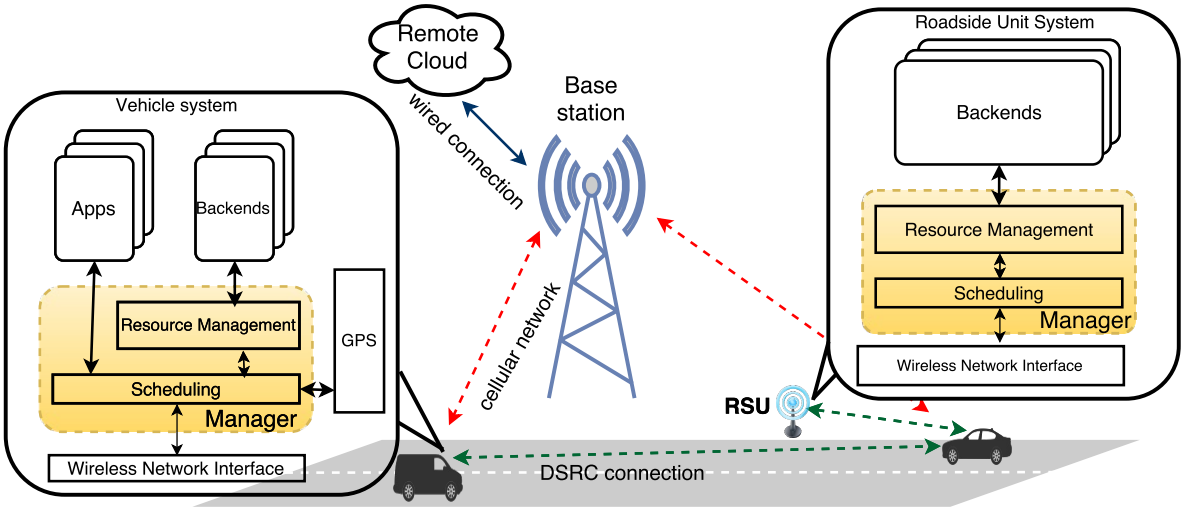


Figure 3.1: System Architecture.

For the job processing model, the remote cloud is assumed to have infinite available computing resources; hence, all offloaded jobs can be processed in parallel. Vehicles and the remote cloud can be connected using cellular networks. However, this cellular connection and the cloud resource are not free; hence, we should reduce their usage as much as possible. Note that the vehicles can also connect to the cloud via the RSU, but this case is not considered here since connections between the RSU and vehicles are used for other purposes. RSUs and vehicles have limited computing capacities, and each node runs a queue to store jobs when their resources are fully occupied. A DSRC standard such as IEEE 802.11p is used to connect the vehicles and RSUs. The DSRC allows a quick link setup without establishing a basic service set, and it reduces the overhead in sending the data to other vehicles.

3.2.2 Job execution

A job is an offloading unit in this framework. The main detail of jobs are discussed in 2.2.3. But the settings are a bit different in this scenario. The difference are stated below.

As we are in a more resource abundant environment, instead of a complex utility function, we let job j have a deadline D_j to be finished before. The deadline is also decided by the application to

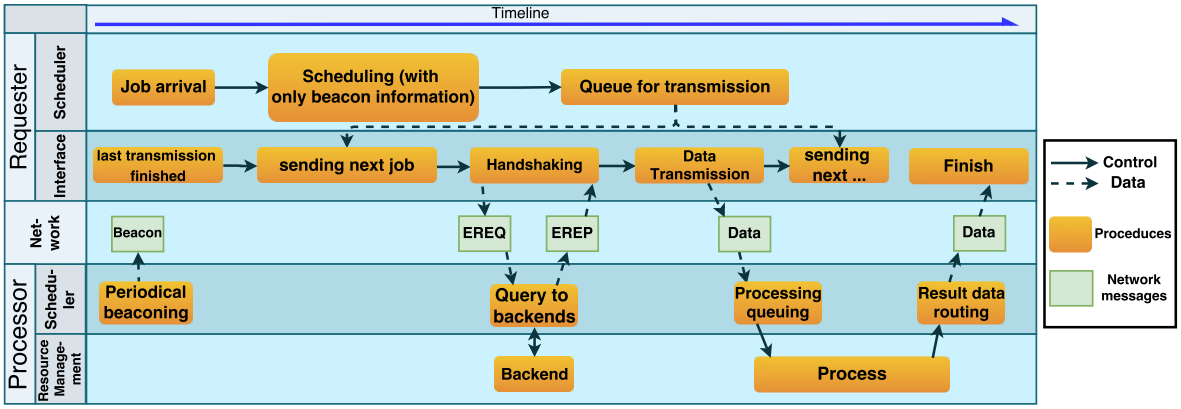


Figure 3.2: Workflow of HVC. Note that we no longer have "flow" concept here. Instead, beaconing, data transmission and scheduling happens in a parallel manner. Job caching and discovery before scheduling are removed.

satisfy the requirements of the job, which may need real-time processing. Jobs are counted as failed if they cannot be finished before the deadline. The proposed scheme attempts to minimize the number of failed jobs using the available resources while reducing the data transmission via the cellular network.

3.3 Workflow

The workflow of HVC is depicted in Fig. 3.2. The procedures in HVC consist of two parts: proactive periodical beaconing and reactive offloading. Some differences can be noticed between this workflow and AVE's (Fig. 2.3).

3.3.1 Beaconing

Similar to the previous work, to help vehicles make scheduling decisions, a beaconing process is included. But in HVC, we want the beacon to carry enough information for scheduling, so that we can cut discovery from the workflow.

By this design, a BM of HVC carries the following information:

- *ID*: The identifier of the source node (vehicle or RSU);

- *Queuing time*: The expected time to finish queued jobs on this node;
- *Location*: Current location of the source node;
- *Velocity*: Speed and direction of the node (for RSU, this field is empty);
- *Processing rate* of the source node;
- *Tags*: As defined in Section 2.2.3, indicating what types of jobs the source node can process.

As beaconing becomes more important in HVC, we shortened the default interval between a vehicle's beacon to 2 seconds in this framework. But it is also configurable for use in different network standards and application scenarios.

Vehicles receiving this message then calculate the ending time of the epoch when the receiver is within the sender's communication range (called "available time"), and store it along with this message. This *available time* is used for scheduling, and the details about its calculation are discussed in Section 3.4. The stored information of the sender expires and is deleted if the corresponding sender's beacon message is not received for a period of time. To allow accidental packet drops, the duration is set as 2 times the beacon interval, i.e., 4 seconds in our settings.

3.3.2 Offloading process

Scheduling

Whenever *manager* module receives a job from the applications, a scheduling process runs immediately to find a *processor* for this job. The caching overhead is thus removed comparing to AVE framework, but also an online scheduling algorithm, i.e. scheduling in real-time without knowing future jobs, is needed.

The job's offloading or local processing will be started right after scheduling, if it is possible. Otherwise when the transmission channel or local resource is currently busy, the job will have to wait in the corresponding queue. In this case, the scheduling process also decides the position of the job in the queue.

The scheduling process aims to process more jobs while reducing the data transmitted through cellular networks. If the scheduling result is denoted as A, the number of finished jobs under this scheduling as $G(A)$, and the total data sizes offloaded through cellular network as $C(A)$, then the objective function can be written as:

$$\text{maximize } G(A) - \alpha C(A)$$

Here, $\alpha \geqslant 0$ is a factor that represents how much disfavor the cellular communication cost would bring. α can be changed according to the different settings.

Scheduling takes input from the stored beacon information and the previously scheduled queues. Same as in AVE, the scheduling problem here is similar to a *2-stage hybrid flow-shop problem*. However as an online algorithm, ant-colony optimization is not efficient. Seeing this, we propose a new algorithm to solve this scheduling problem, which is presented in Section 3.4.

Handshaking and transmission

Offloading the scheduled jobs to the remote cloud via a cellular network is trivial, and the scheduled jobs can be uploaded one by one. However, regarding DSRC offloading (to RSU or neighbor vehicles), the situation is more complicated due to the rapidly mutating vehicular environment.

In AVE, this problem was solved by a complete discovery right before transmission. But here we have only part of the information from beacons when scheduling is done. That brings more unexpected factors when the job is being offloaded. To provide reliable offloading, a "handshaking" procedure is introduced in HVC. Whenever a requester is attempting to offload a job to its assigned processor, it first sends out a *hybrid request message* (HREQ) to the processor. The information carried in a HREQ is composed of the following: (a) ID of the requester, and (b) the scheduled start time of the job, which will be mentioned later. Then, it waits for a short duration of time (default to 50 ms, which is the length of a SCH access interval in IEEE 802.11p[77]). If no response is received until the end of this duration, the requester assumes that the connection to the processor is lost. The requester then removes the processor node from its stored list. All future jobs assigned to this processor, including

the one currently being sent, will be rescheduled in the order of their arrival. If the processor receives this HREQ, it checks its processing queue. If the queuing time is less than the scheduled start time attached in the HREQ, processor sends back an "accept" message to the requester, indicating that the offloading is feasible. The job's data transmission starts immediately after. Otherwise, if the current queuing time is greater than the scheduled start time indicated in the HREQ (due to new jobs' arrival since last beacon), processor sends back a "decline" message. The updated queuing time is also attached in the message. When the requester receives this message, it checks its DSRC queue, picks out all jobs that are scheduled to be processed in this processor before the new queuing time, including the one that is pending for transmission, and reschedules them. After processing, processor sends the result data directly back to the requester. This can be done in the same way as how the offloading data were transmitted, and if the DSRC connection is lost, the result will have to be transmitted through a cellular network.

3.4 Scheduling algorithms

This section introduces how to calculate the available time and the scheduling algorithm.

3.4.1 Available time

As described in Section 3.3, when a vehicle receives a beacon message from an RSU or neighbor vehicle, it calculates the "available time" of the corresponding node, which indicates the expected time instant till when the sender stays within its communication range.

If the beacon sender is an RSU, the vehicle calculates the available time as:

$$t_{available} = t_0 + \frac{2\mathbf{v_0} \cdot \mathbf{p_0}}{\|\mathbf{v_0}\|^2}$$

Here, t_0 is the time when the vehicle first receives a beacon from the RSU, $\mathbf{v_0}$ is the velocity vector of the vehicle at t_0, and $\mathbf{p_0}$ is the position vector of the RSU relative to the vehicle at t_0. Occasionally, $\mathbf{v_0} \cdot \mathbf{p_0} \leqslant 0$, and this means that the vehicle is driving away from the RSU at t_0. In this case, $t_{available}$

is set as t_0, which means that the RSU is not considered for future jobs due to the unstable link. The rationale behind this formula is that the vehicle assumes that the coverage area of the RSU is a circle, and the first beacon indicates that it is driving into such a circle. If the velocity remains the same, the distance that it will travel in the circle is:

$$2\|\mathbf{p_0}\|(\mathbf{v_0} \cdot \mathbf{p_0})/\|\mathbf{v_0}\|\|\mathbf{p_0}\|$$

Therefore, at $t_{available}$, as calculated above, it will be driving out of this circle.

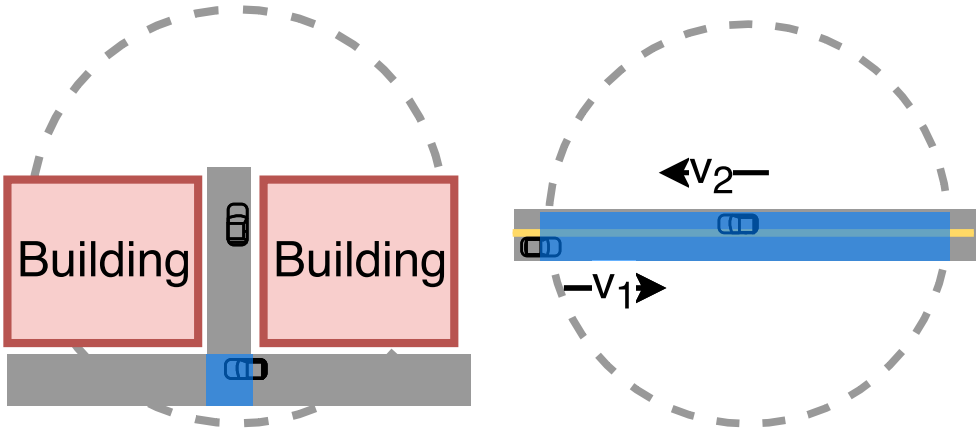
The available time between two vehicles in an urban area is calculated as follows:

(i) If they are driving on different roads, $t_{available}$ is also set as t_0. Moreover, the expected duration of their link is too short in this case, as shown in Fig. 3.3a.

(ii) If they are driving on the same road but in opposite directions, $t_{available}$ is calculated based on the relative speed $\mathbf{v_1} - \mathbf{v_2}$, as shown in Fig. 3.3b.

(iii) If they are driving in the same direction on the same road, the time calculation is trivial and can refer to (ii)), where the vehicles drive on the same road but toward different directions. Because both vehicles' movement is restricted by the road, the time calculation in (ii)) can be further extended to calculate the time when they move into different roads. Assuming that vehicle 1 is l_1 meters away from the next junction, and assuming l_2 for vehicle 2, the time that these two vehicles are on the same road can be written as $min(l_1/\|\mathbf{v_1}\|, l_2/\|\mathbf{v_2}\|)$. The available time in this case is the minimum between the result value from (ii)) and $t_0 + min(l_1/\|\mathbf{v_1}\|, l_2/\|\mathbf{v_2}\|)$.

The calculated result is stored along with other information of the sender (processing rate, *tags*, and so forth) and used in future scheduling. Although there are some assumptions in this calculation, which occasionally might not be accurate in the real world, this still reduces erroneous assignments where jobs are assigned to a node that will not be able to reached at the scheduled time. As the handshaking procedure exists, the harm of an inaccurate estimation of the available time is reduced. Therefore, such a rough but simple estimation is accepted.



(a) When two vehicles are driving on different roads, as buildings block signals, the link duration is very short.

(b) When two vehicles are driving on the same road but in opposite directions, the link duration depends on the relative speed.

Figure 3.3: Available time between two vehicles. The dashed circle indicates the communication range of a vehicle, and the blue areas are where the other vehicle would have direct DSRC connection with it.

3.4.2 Job scheduling

The job scheduling algorithm finds a processing node for each job immediately upon its arrival, and it also specifies the queuing position if there are other jobs waiting to be processed. The inputs of the scheduling algorithm are as follows: P is the set of RSUs and vehicle nodes that can satisfy the job requirements, including the ones from stored beacon information, and the requester itself. For each node i, we know its processing rate r_i, finishing time of current processing queue q_i and "available time" b_i. The offloading (uploading) time of a job j through the DSRC channel is estimated according to its data size and DSRC transmission rate, and it is denoted as tu_j. The result returning (downloading) time is td_j. The processing time for job j on node i is estimated as tp_{ij} according to the job's length and the node's processing rate. j' denotes the job that is going to be assigned, and the deadline of j' is denoted as $d_{j'}$.

Unless data transmission and job processing are all finished before the arrival of the next job, we will maintain queues of the jobs, which are waiting for local processing, DSRC transmission and/or cellular transmission. We denote these three queues as Q_{proc}, Q_{DSRC} and $Q_{cellular}$, respectively. The node

assigned for a job j in a queue is denoted as $p(j)$. Each job in queues also receives a "scheduled start time" (SS_j). The first job in queues Q_{DSRC} and $Q_{cellular}$ is the job being transmitted; for convenience, the scheduled start time of this job is set as the time when it finishes transmission. Similarly, the first job in Q_{proc} is the job being processed, and its SS time is set as when the job processing starts. How to set SS for the remaining jobs is shown in Algorithm 3. The "scheduled start time" can be interpreted as "the latest time allowed to start processing the job". If processing cannot start before this scheduled start time (due to unexpected events such as disconnection or other reasons), the scheduling result becomes invalidated. Therefore, a handshaking procedure is employed for confirmation, as discussed in Section 3.3.

The general idea of this algorithm is to greedily search an "empty slot" for transmitting and an "empty slot" for processing the coming job upon its arrival. Further adjustment of the scheduling results is made for better performance. The empty slot ensures that the duration from SS_j to $SS_{next(j)} - tu_{next(j)}$ is free for transmission, where $next(j)$ is the next job of j in the queue. Moreover, the assigned node should be available, i.e., no other jobs are scheduled to be processed from the time it receives all the data to SS_{j_2}, where j_2 is the job being scheduled to the same node and is located after j in the queue. When we find the "empty slot" for processing, the algorithm searches for a length of at least $tp_{ij'} + td_{j'}$ rather than the actual processing time $tp_{ij'}$. The reason for this is that we cannot control the transmission of the results, which is initiated by the other node. Therefore, we use a larger duration to better guarantee the transmission of the returned results. In actual processing, this extra time margin will be eliminated.

The algorithm scans the queues from back to front. If there are multiple such slots, the one with the latest starting time will be selected such that future jobs can find the suitable slots easier. The search process does not delay scheduled jobs. Hence, no job will be scheduled to a place where it cannot catch up on its deadline. If all attempts to find such slot fail, the algorithm places j' at the end of $Q_{cellular}$ for offloading to the remote cloud. The detailed algorithm is explained in Algorithm 3.

For each job, at most $|P|$ iterations are needed, and in each iteration, each job in the queues is

checked at most twice. Let the numbers of these jobs be n and $m = |P|$; then, the time complexity of assigning one job is $O(nm)$, which is trivial.

3.5 Evaluation

This section presents the simulation setup and the performance evaluation of the proposed scheme in comparison to competing schemes with different parameters settings.

3.5.1 Simulation setup

The general simulation stack is same as previous work, shown in Fig. 2.8, that we use *Omnet++*, *Veins*, *SUMO* and *LuST scenario* for the evaluation.

But in this scenario, we only choose a part of the urban area for the simulation. The selected area is shown in Fig. 3.4. Two RSUs are deployed in the selected region, and the locations of the RSUs are marked with blue points in the figure. We choose 15 minutes from 6:00am to 6:15am as the evaluation time range, during which the traffic is in a median range, neither the highest nor the lowest.

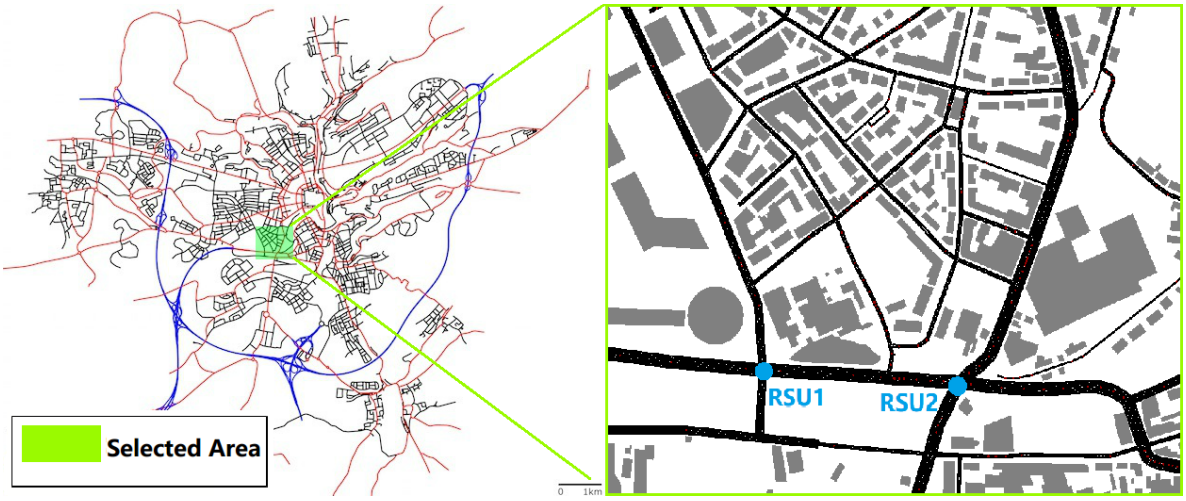


Figure 3.4: Map used in this evaluation. The left part shows the global map, and the green rectangle in the center shows the location of the simulated scenario; The right part shows the detailed map – gray boxes are buildings (obstacles) and the locations of the RSUs are marked with blue points. Part of the figure is from *LuST*[71]

The commonly known IEEE 802.11p DSRC standard is also used here for the connections between vehicles. Parameter settings are the same with previous work, which is introduced in Section 2.5. For simplicity, RSUs use the same channel with vehicles.

To simulate the cellular network, the cellular connection for each vehicle is treated as an ever-connected wired network. Each vehicle, disregarding where it is within the scenario and how many users are around, has an individual 5 MByte per second cellular transmission rate. The link setup (RRC negotiation, DNS look-up, querying cloud server, and so forth) and the transmission between the remote cloud through cellular networks for upload or download introduces a 100 millisecond delay when the cellular interface is waken from idle. Note that the performance of the cellular network is typically worse than this assumption in the real world due to network sharing. Because cellular usage is what the proposed framework attempts to reduce, we intentionally let cellular network to be ideal, so that the evaluation can show how our proposal competes with the upper bounds of existing cellular offloading schemes.

Note that our scheme is not sensitive to these parameters and that these parameters vary across different countries (other factors can also affect the cellular performance, such as the DNS process); thus, we use these parameters to show the performance of the proposed scheme in this evaluation.

We assume the following scenario when a vehicle moves toward its destination. During driving, the driver or the passengers activate some application and jobs are generated periodically, e.g., the passengers activate an AR application to look for interesting nearby objects and shut down this application when the request is satisfied. Such types of applications are selected based on personal preference; hence, it is quite independent between different users. When the application is activated, requests (jobs) are generated. Then, it requests the framework to process the jobs.

We use a *discrete-time 2-state Markov chain* to model this process, which is shown in Fig. 3.5. In this model, each application has 2 states: *idle* and *busy*. In the *idle* state, the application is placed in the background and does not generate jobs. In the *busy* state, jobs are assumed to be generated following a *Poisson process*, with an arrival rate of r (r=2.0 by default) jobs per second. Transitions between the two states are with probability p and q per second. In our evaluation, p, i.e., the probability

of transitioning from *busy* to *idle*, is set to 0.1 by default. In this way, the expected sojourn time in the *busy* state is 10 seconds.

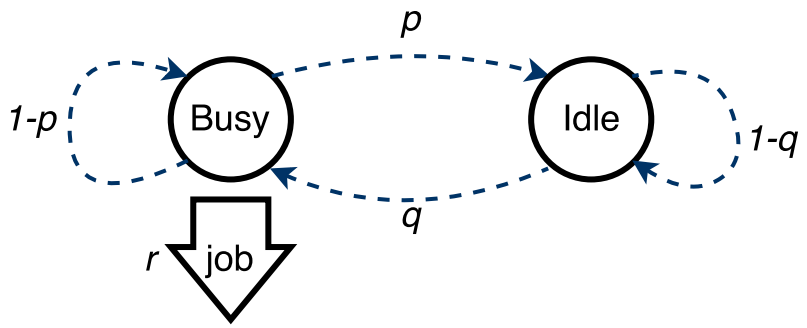
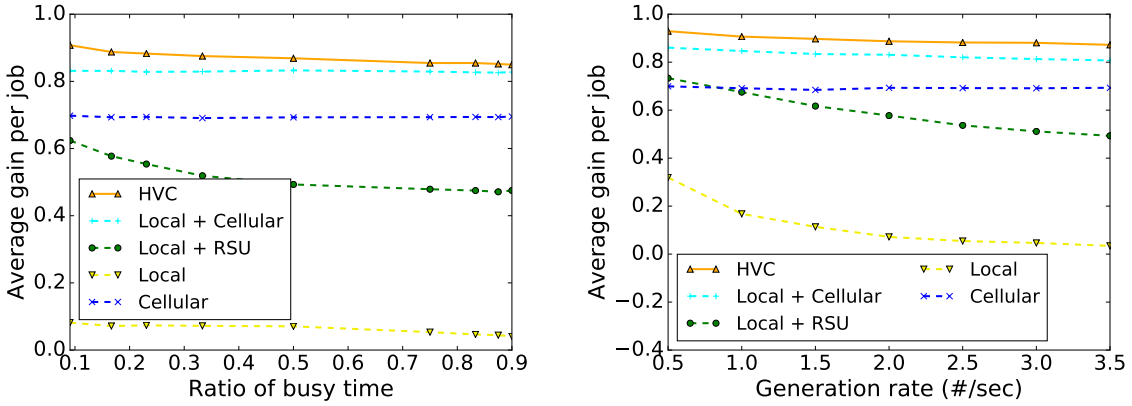


Figure 3.5: Illustration of the 2-state Markov chain model.

The *tags* (defined in Section 3.2) illustrate the host requirements. We assume there are 7 tags in total. Each vehicle has 4 tags (i.e. generating 4 kind of jobs that have different compatibility demands), and the tags are selected from the 7 tags randomly. For a fair comparison with other schemes, each vehicle is guaranteed to have the corresponding backends (tags) for its generated jobs in this simulation.

The lengths of jobs follow an exponential distribution with a mean of 4.0 seconds by default. The deadline of each job is set as 1.5 to 2.0 times its *length*, counting from when the job is generated. For simplicity, the size of offloaded data and the size of the returned results are set to be the same. The sizes also follow an exponential distribution, and the mean is 100 KByte by default. The processing rates are set as 1.0 to 2.0 for each vehicle. RSUs have a higher processing rate, which is set as 5.0. Remote clouds are considered to be significantly faster with their powerful servers; therefore, the processing rate is set as 20.0. As previously mentioned, both vehicles and RSUs are only able to process one job at a time, whereas remote clouds have infinite computing capacities and can process all jobs in parallel. The value of α, which is used to represent the disfavor of cellular data usage cost, is set to be 1.5 unit per megabyte. Note that other values of α also work in our scheme, and we use this as an example.



(a) Under different ratios of busy time.

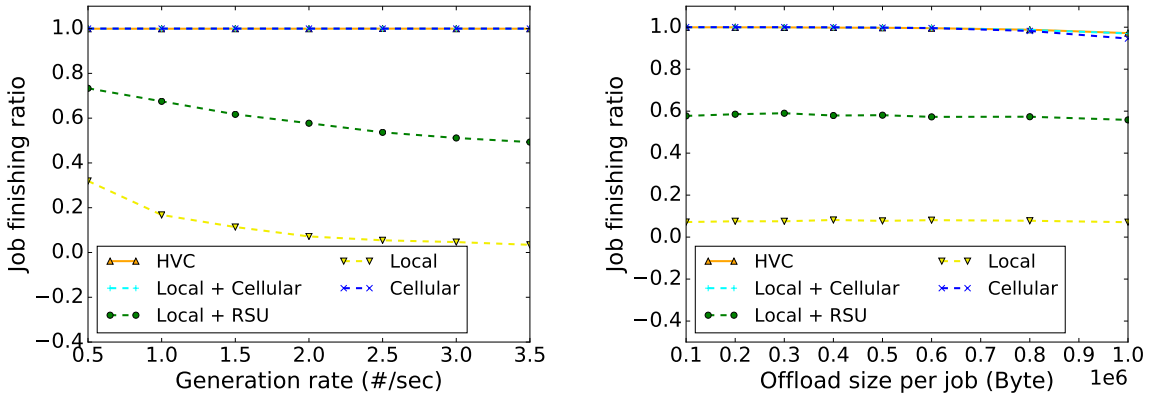
(b) Under different job generation rates.

Figure 3.6: Average gain per job in different settings of HVC.

3.5.2 Simulation results

To verify the performance of HVC, we choose four competing schemes: 1. *Local* – there is no offloading, and each vehicle has to process jobs by itself; 2. *Cellular* – all jobs are offloaded to the remote cloud via a cellular network; 3. *Local+Cellular* – jobs can choose between *Local* and *Cellular*; 4. *Local+RSU* – jobs can either be offloaded to RSU when available or processed locally. The online algorithm proposed in Section 3.4 is used in *Local+Cellular* and *Local+RSU* for scheduling between local processing and other options. *Local* is what we have in the vehicular environment right now, and there is no offloading framework. The *Cellular* and *Local+Cellular* schemes are traditional cloud computing methods, whereas the latter also assumes that the local system has some job processing capability. *Local+RSU* resembles the *Cloudlet*[78][79] technology. We run the simulation 5 times for each set of parameters, and the average values are calculated as the results shown in the figures.

We first evaluate the performances of these schemes under different frequencies of "activating" the applications, i.e., transition from the *idle* state to the *busy* state. As we know, the ratio of the expected sojourn time in the *busy* state to the total time is $q/(p+q)$. The average gain per generated job is evaluated under different values of $q/(p+q)$, and the results are shown in Fig. 3.6a. As shown in this figure, *Local* has the lowest outcome because the workload (with deadline) is far beyond the



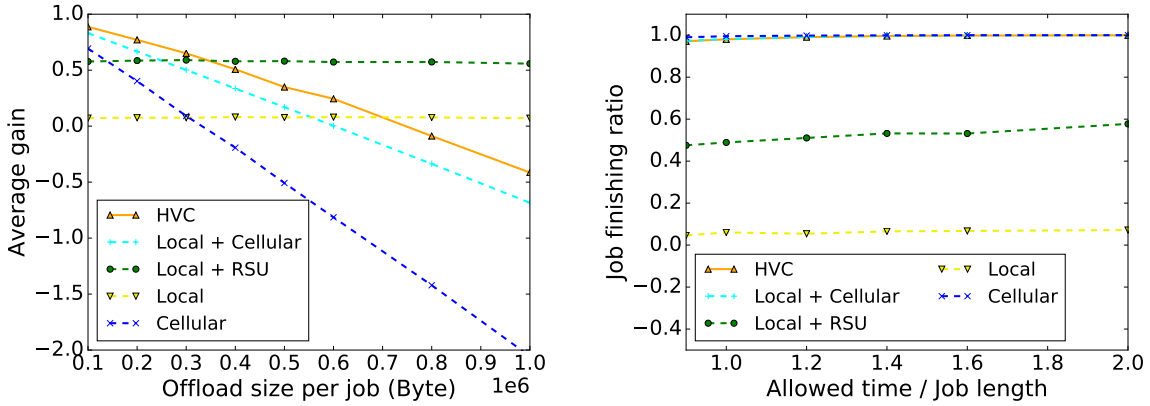
(a) Under different job generation rates.

(b) With different sizes of jobs.

Figure 3.7: Job finishing ratio in different settings of HVC.

vehicle's computing capability. When RSUs are introduced, this situation is improved (the results of *Local+RSU*), but the performance is limited by the RSUs' finite resources and the DSRC channels. The two competing schemes *Cellular* and *Local+Cellular*, which have cellular connections with or without local capability, have generally better performance, and this is due to the ideal assumption of the cellular network and the unlimited resources in the remote cloud. Both of these schemes do not suffer from the increasing usage of the applications. However, due to the cost of data transmission using cellular networks, a cellular network penalty on gain is present. The proposed scheme *HVC* has the best outcome among all the competing schemes since it can efficiently use the nearby vehicles and RSUs. However, because the available computation capacity from the neighboring vehicles and RSUs is limited, *HVC* has to rely on cellular networks for excessive workloads, which decreases its gain when there are more busy states.

In Fig. 3.6b and Fig. 3.7a, the results with varying job generation rates r are shown. In contrast to the value of q, increasing r does not mean that more users are busy simultaneously. It only increases the workloads of the busy vehicles, and increases the workload variance among vehicles. Therefore, as shown in Fig. 3.6b, *HVC* outperforms the other competing schemes at higher generation rates due to the way of efficiently using the idle neighbors to help the busy ones. Regarding *HVC* and



(a) Average gain per job with different sizes of jobs.

(b) Average job finishing ratio with different deadlines.

Figure 3.8: Impact of data size and deadline of jobs in HVC.

Local+Cellular, the jobs that cannot be processed locally or via the neighboring vehicles and RSU will be uploaded to the remote cloud via the cellular networks. Due to the powerful remote cloud servers and the assumed ideal cellular network, these jobs can be well processed by the remote server. Hence, we can observe that the finish ratio of the *HVC*, *Cellular* and *Local+Cellular* schemes all achieve a 100% finishing ratio from Fig. 3.7a. However, *HVC* relies much less on the non-free cellular network and remote cloud resource.

Then, we evaluate the impact of the offloaded data's size. The results are shown in Fig. 3.7b. When the size is large enough (over 800 KByte one-way per job), *HVC*, *Cellular* and *Local+Cellular* cannot maintain a 100% finishing ratio due to the bandwidth limitation of the cellular link. However, *HVC* performs better than *Cellular* and *Local+Cellular* because of the multi-sourcing characteristic. Conversely, if the cellular cost is considered, the value of the objective function can be calculated, and the results are shown in Fig. 3.8a. A main reason for why the gain of *HVC* falls quickly is that in the current settings, the framework would attempt to offload jobs with data sizes that are too large to the cellular network, even though abandoning it might lead to a higher value of the objective function. In other words, the gain from processing the job is smaller than the cost of the cellular usage. However, in some scenarios, where one job's finishing relies on other jobs' successful processing (dependency

across jobs exists), this is a necessary method to guarantee the finishing of consequent jobs.

The ability of this framework in terms of processing time-sensitive jobs, i.e., jobs with tight deadlines, is also investigated. We vary each job's deadline from 0.9 to 2.0, i.e., each job's deadline will be 0.9 to 2.0 times its *length*. The results are presented in Fig. 3.8b. With tighter deadlines, e.g., jobs must be finished within $0.9length$ time, the overhead of *HVC* reduces the finishing ratio slightly; this case occurs when the handshaking process fails and jobs have to be rescheduled. Generally, *HVC* has similar performance with *Cellular* and *Local+Cellular*, but *HVC* consumes considerably less cellular network resources.

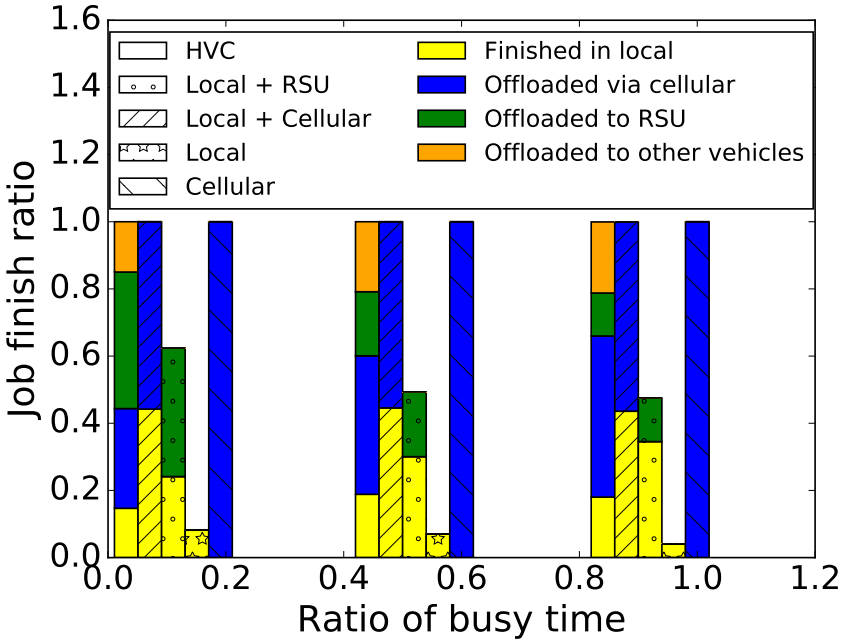


Figure 3.9: Average job finishing ratio and the shares of each type of nodes under different ratios of busy time.

The results in terms of the job finishing ratio are presented in Fig. 3.9. We can observe that even with the highest busy state frequency (90% of a vehicle's time is in the busy state), the schemes with the cellular option (*Cellular*, *Local+Cellular*, and *HVC*) can finish 100% of jobs before their deadlines due to the powerful remote cloud servers and the cellular networks. The difference between these three

schemes lies in the size of the data transmitted via cellular networks. *HVC* utilizes RSUs and neighbor vehicles to process more jobs. However, less jobs are processed locally because vehicles may have to process others vehicles' jobs. From a global perspective, this type of resource pooling helps to reduce the average cellular usage.

Moreover, same as AVE's evaluation, the simulation conducted here does not take small scale fading of DSRC communication into consideration. Therefore in real environment, all schemes other than *Local* and *Cellular* may have to offload more jobs with cellular network, or process more jobs with local resources. On the other hand, we can also view a worse transmission rate as "more data to send". As we can see in Fig. 3.7b and Fig. 3.8a, if the transmission rate is lowered, finishing ratio and gain will drop accordingly. And in Fig. 3.9, more jobs may be finished locally or offloaded via cellular, and as Doppler effect is severer in V2I transmission, RSU's portion may decrease too.

3.6 Limitation on communication and solution with millimeter wave

In the evaluation done in Section 3.5, to show how competitive the proposed scheme is against traditional offloading with cellular connection, we assumed a very ideal cellular network, where each connection is independent and has an exclusive channel. But the real cellular network is not ideal. End-users belongs to the same base station have to share fixed amount of resources.

In Fig. 3.10, we do a simple simulation to estimate the impact of shared cellular channels. We divide the urban scenario used in Section 3.5 into $200m \times 200m$ blocks, and put a total bandwidth limit in each of the blocks, *i.e.* all vehicles within this block will have to share that bandwidth. Note that this is not an accurate evaluation, in which we must consider difference size of cells, distance to base station, and the bandwidth consumed by non-vehicle users. Also we do not take pedestrians' consumption on cellular network into account, assuming they are using another isolated portion. This simulation result serves as a demonstration about the performance of the framework in a not-so-ideal cellular network.

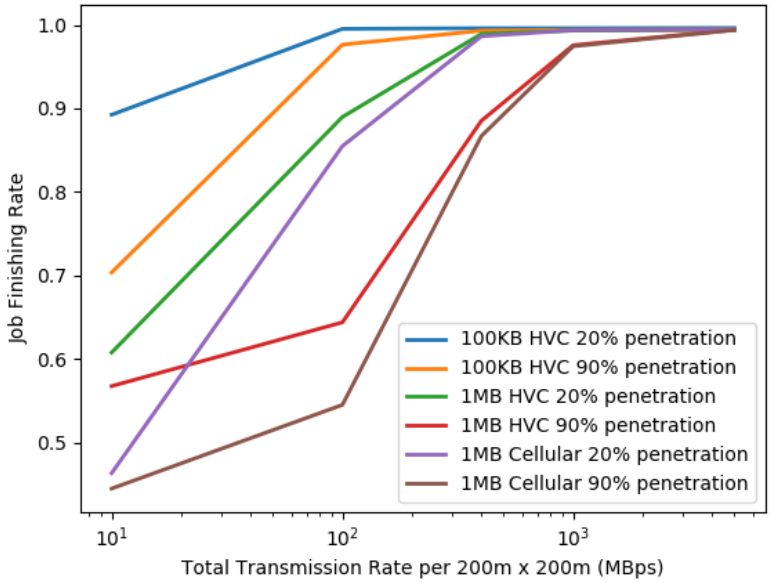


Figure 3.10: Job finishing ratio under difference cellular settings. In the legend, $100KB/1MB$ specifies the average data size of jobs; HVC/Cellular indicates whether the whole HVC framework is applied, or just with cellular offloading (i.e. same as traditional); the last percentage shows the penetration rate setting, higher penetration rate means more vehicle users.

We use the default parameters provided in Section 3.5, and also add other cases. As we can see in the figure, *HVC* schemes, which is our proposed scheme, generally outperform pure cellular offloading schemes. And we also notice that, in higher penetration rate, i.e. more vehicles participating, all schemes suffer from the lowered performance of cellular offloading. Apart from vehicle density, the size of jobs also has significant impact on the performance.

On the other hand, we also notice the limitation on data rate with current vehicular DSRC standards. Take IEEE 802.11p for example, the theoretical upper bound is $27Mbps$. In Fig. 3.7b of Section 3.5 we can see larger jobs suffer from transmission delay in DSRC network. This motivates us to find a faster communication method, for such kind of data intensive jobs.

There are multiple communication methods that can provide high bandwidth. For that purpose, they are operating on extremely high frequencies. Examples are like millimeter wave[80], infrared

communication[81] and visible light communication[82]. Among them [80] and [82] are also applied in vehicular environment. They share the characteristic that while having high transmission rate, they mostly require light of sight in transmission.

Millimeter wave (mmWave) is one of the options we are interested in. Utilizing frequency above $30\,GHz$, this technology enables transmission rate up to $10\,Gbps$ [83]. IEEE 802.11ad with mmWave spectrum also has up to $6.76\,Gbps$ transmission rate[84]. And when compared with visible light technology, it is less sensitive to change in relative position between sender and receiver, which gives it advantage in dynamic vehicular environment. This technology is promising in forming the next generation of mobile networks[85].

Like other options, this communication technique suffers greatly from obstacles. The 1 to 10mm wavelength means it will be blocked by most visible objects[86], thus requires line of sight between both ends. To overcome this difficulty, we propose that instead of using only direct connection, the pair can also communicate with each other with up to 1 relay in between. In vehicular environment, the relay can be the vehicle with the highest antenna, or on traffic lights[87][88], RSUs, and light poles.

Note that relay here is different from *forwarder* in the AVE framework. It does not need to have the proposed framework installed but only functions as a repeater for millimeter wave. Therefore it can be made very cheap and deployed everywhere, apart from millimeter wave interfaces on vehicles.

For simplicity in the following simulation, we assume that this 2-hop method achieves 90% success rate. The framework is modified to adapt to the new transmission method. 802.11p is still used for beaconing as directional millimeter wave is not good at broadcasting. And in scheduling, where it cannot be certain if a millimeter wave connection will be available when the job is being transmitted, jobs are scheduled assuming they can be transmitted with millimeter wave. Then when millimeter wave fails in actual transmission, the framework choose from (a) retrying transmission with 802.11p, if possible; or (b) rescheduled the job if 802.11p cannot make it in time. Other parts of the framework remains the same. We run the simulation under the same parameter settings as Section 3.5, except for the data volume of jobs being changed to $1\,MByte$.

The results are shown in Fig. 3.11. The relative number of jobs finished is calculated by dividing

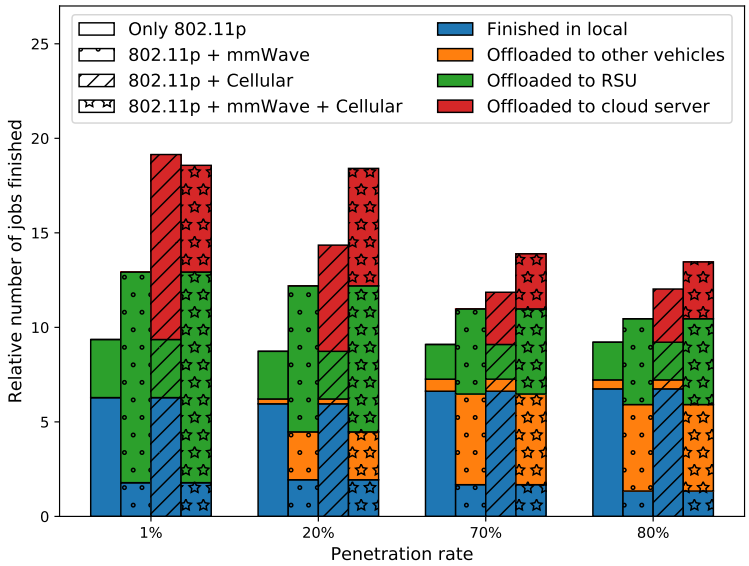


Figure 3.11: Number of jobs finished with different combination of 802.11p, millimeter wave and cellular network schemes.

each scheme's number of finished jobs with the number of jobs that can be finished without any sort of offloading, so that the differences on number of vehicles and generation of jobs are canceled. We can see that the addition of millimeter wave significantly improves schemes that utilize only 802.11p. More jobs are allowed to be offloaded with the fast connection, reducing the number of jobs forced to run locally. It also helps in schemes with cellular offloading, as it removes burden from cellular network in crowded scenarios.

However, we also notice that when the penetration rate is extremely low, schemes with both millimeter wave and cellular offloading has lower performance than the one with millimeter wave. One reason for this phenomenon is the failure chance in millimeter wave communication. In the current framework, failed millimeter wave transmission adds overhead in rescheduling, thus makes jobs with tight deadline unable to finish. We consider this problem solvable with better scheduling methods, taking the success rate of millimeter wave into account.

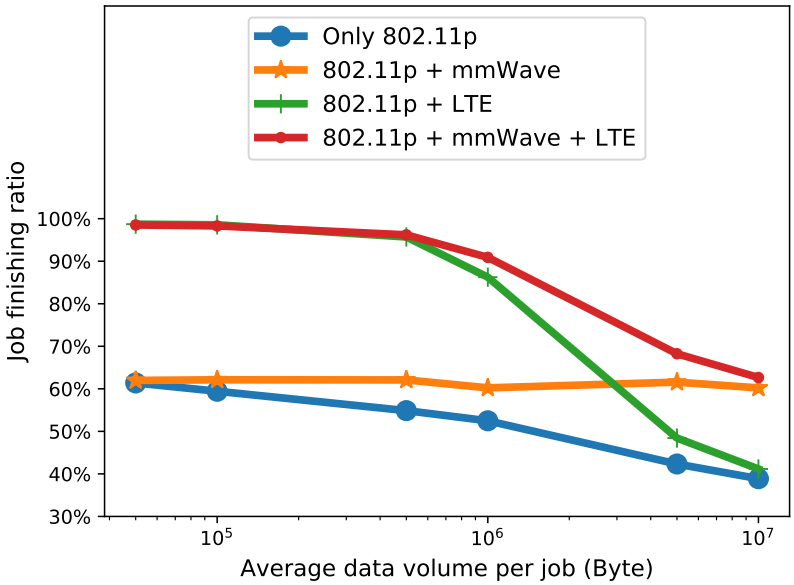


Figure 3.12: Ratio of jobs finished with different data volumes, with combination of 802.11p, millimeter wave, and cellular schemes.

When it comes to larger jobs, as it is seen in Fig. 3.12, the advantage of millimeter wave communication is more significant. Schemes with millimeter wave option retain finishing ratios above 60% even for jobs as large as $100MByte$, while other schemes fall under 40%.

Without a widely accepted millimeter wave communication standard in vehicular environment, this evaluation is just a rough estimation of its potential. From the results we can tell a faster network standard helps greatly in vehicular cloud computing. However, the result also suggests that current scheduling algorithm does not completely exploit the benefits from combination of 802.11p and millimeter wave. Thus how to efficiently utilize this kind of faster but more unreliable connections, might emerge as a new problem when they are standardized.

3.7 Summary

In this chapter a framework named Hybrid Vehicular Cloud/Edge Computing (HVC) for cloud computing on the road is proposed. The proposed scheme can be used as a general framework for all types of applications; is able to satisfy various types of job requirements, such as real time or host requirements; and is adaptive to the dynamic vehicular environment.

Also, an online scheduling algorithm is proposed. It allows more jobs to be processed while reducing data transmission via non-free cellular networks. Extensive simulations are conducted, and the simulation results show the superiority of the proposed scheme over competing schemes in typical scenarios.

Simulations are also done to show the limitation in communication with actual cellular network and 802.11p DSRC standard. An alternative method with millimeter wave transmission is evaluated. The result shows such a fast transmission standard will help significantly in vehicular cloud computing, despite of its drawback in reachability.

Algorithm 3: Scheduling Algorithm of HVC

```
   // Quick assigning to idle nodes
 1 if Q_proc is empty then
 2 |   Assign j' to local processing and set SS_j' to current time;
 3 |   return;
 4 else if Q_DSRC is empty then
 5 |   Assign j' to the fastest neighbor node and set SS_j' to t_0 + tu_j', where t_0 is current time;
 6 |   return;
 7 end
   // Searching
 8 while P' ≠ ∅ do
 9 |   Find the node i with the highest r_i, remove i from P;
10 |   if i is not the requester then
11 |   |   Search from the back of Q_DSRC to find jobs j, j_1, and j_2 that satisfy p(j_1) = p(j_2) = i,
   |   |     {j̄ | p(j̄) = i and j̄ is between j_1 and j_2} = ∅,
   |   |     SS_j + tu_j' + tu_next(j) ⩽ min(SS_next(j), b_i, d_j'),
   |   |     SS_j1 + tp_ij1 + td_j1 + tp_ij' + td_j' ⩽ min(SS_j2, b_i, d_j'), SS_next(j) ⩽ SS_j1, and
   |   |     SS_j + tu_j' + tp_ij' + td_j' ⩽ min(SS_j2, b_i, d_j') ;
12 |   |   if found then
13 |   |   |   Assign j' to node i and insert it to Q_DSRC right after j;
14 |   |   |   Set SS_j' to min(min(d_j', b_i, SS_j2) − tu_j' − tp_ij', SS_next(j));
15 |   |   end
16 |   else
17 |   |   Search from the back of Q_proc until a job j is found that satisfies
   |   |     SS_j + tp_ij + tp_ij' ⩽ min(SS_next(j), d_j, b_i);
18 |   |   if found then
19 |   |   |   Assign j' to node i and insert it to Q_proc right after j;
20 |   |   |   Set SS_j' to min(SS_next(j), d_j, b_i) − tp_ij';
21 |   |   else if the time to finish current Q_proc ⩽ min(d_j, b_i) − tp_ij' then
22 |   |   |   x ← finishing time of currently processing job;
23 |   |   |   for each job j from the second one in Q_proc to the last do
24 |   |   |   |   SS_j ← x;
25 |   |   |   |   x ← x + tp_ij;
26 |   |   |   end
27 |   |   |   Assign j' to node i and insert it to Q_proc's end;
28 |   |   |   Set SS_j' to min(d_j, b_i) − tp_ij';
29 |   |   end
30 |   end
31 end
   // Cellular offloading as last result
32 if all searching above fails then
33 |   Add the job to the back of Q_cellular;
34 end
```

Chapter 4

Discussion and Conclusion

4.1 Simulation and network issues

Although we try our best to simulate real world network, there are still differences. In this section we discuss how those possible differences may affect the actual performance of the framework.

4.1.1 Small scale fading

Due to restriction on the simulation software and computational power, small scale fading of radio signal is not well simulated.

One notable effect is Doppler shift. Vehicles are moving faster than hand-held devices, hence having stronger Doppler effect. With this in consideration, 802.11p standard doubles symbol duration and guard interval. But it may still cause extra bit error in actual road environment, especially for V2I communications, where the relative speed is higher. In our simulation, this effect is not considered. Hence compared with simulation, transmission to infrastructures can be slightly worse in real world.

Multipath interference is another issue. Radio signal is reflected and diffracted by objects in the world, forming multiple transmission paths with different length and difference propagation time. This interference also introduces bit errors in transmission. Again, due to computation complexity, it is not

simulated in our framework. This means real world cases in urban area, where objects are much more than rural roads or highways, may not be as good as simulated results.

4.1.2 Network congestion

In our simulation, only communication for our proposed framework is considered. But when smart vehicles become popular in the future, more applications will share the limited channels of VANET. Therefore the available bandwidth for proposed framework may decrease. Offloading will be harder in such case, and the performance of both frameworks will be worse.

Moreover, with more transmission happening in the channels, failure rate of packets increases. This is not a problem for data transmission or discovery procedures in the proposed frameworks, as they can implement retransmission mechanisms. But it may cause beacon messages, which is sent once in each interval and forgot, to delay or fail, and lowers the performance. In such scenario, AVE is better than HVC in that AVE does not rely heavily on beaconing, and insensitive to delays on beacon messages.

However, even in the worse case scenario where network is overloaded and offloading is near impossible, the proposed frameworks should at least have nearly same performance as schemes without any offloading. With poor VANET condition and offloading failure in mind, the frameworks are designed with low overhead and fallback methods, e.g. skipping discovery according to NAI value in AVE.

Also, as universal cloud computing frameworks, the proposals can aggregate transmissions of cloud applications, e.g. sending only one beacon message for numerous applications in the system. In this way the proposed frameworks reduce the chance to congest the VANET.

4.2 Other discussions

In this section, some aspects that are not the focus of the author's work, but might help in different application scenarios, are discussed.

4.2.1 Centralized management of vehicular cloud

The proposal in this thesis can be applied to some other scenarios.

A notable one is to consider the introduction of a centralized management layer. If the proposed cloud computing scheme becomes popular, governments and companies would then like to build a platform to monitor, manage and profit from the system. Therefore, some sort of centralized management can be anticipated in the future. With a centralized management, the discovery phase and scheduling algorithms can be run in the cloud server instead of each vehicle, hence better performance can possibly be achieved. However, under such condition, the inherent problems of vehicular network still exist: limited bandwidth, unstable connection and more importantly, the coverage of centralized controlling infrastructures. Considering this, a combination of decentralized scheme, as described in this thesis, and traditional centralized scheme should be used. How to efficiently hand off between the two scheme and utilize centralized resources would be interesting problems.

While the thesis does not focus on the centralized part. The scheme can be extended with centralized management.

A plain-forward way is to form a centralized cloud computing environment in the coverage area of centralized managed infrastructures. And for remote areas without such coverage, we apply AVE framework or HVC framework, depending on what infrastructures are available. When a vehicle is driving from one framework's coverage to another, we can do the following hand-off procedures:

- From AVE to HVC and backward. The main difference between AVE and HVC is in scheduling and offloading workflow. When a vehicle transits from AVE to HVC framework, it just has to immediately clear its job queue by assigning them one by one in HVC's style, and switch the beaconing to adapt to HVC's beacons. On the other hand, from HVC to AVE is simpler, in that the vehicle would just start caching jobs. But if there are jobs scheduled to remote cloud and the result has not yet returned while the vehicle is leaving cellular coverage, we may have to redo the offloading for that job in new framework.

- From AVE/HVC to centralized management. The vehicles from decentralized frameworks can

finish their own workflow for the jobs that have been cached/scheduled/offloaded. And for new jobs arriving, the vehicles can then start using centralized resources to help.

- Leaving centralized management. The vehicle would have to abandon outdated information from centralized server. If there are jobs being scheduled in centralized servers, the vehicle may have to redo the scheduling again, just like from HVC to AVE.

Indeed, there will be more problems in detail implementation in such hybrid environment. Here we are just going to give an overview of the possibility of our proposed framework here. The author thinks these ideas can be the incentives for the next better vehicular cloud computing solutions.

4.2.2 Alternative scheduling problems

In realistic applications, scheduling problems can be different from what we assume in AVE and HVC. In this case, the scheduling algorithm needs some modifications to solve the new problems.

A reasonable alternative to the ACO-based algorithm in AVE is to consider "hard deadlines" on jobs, like the one in HVC's assumption. For this we can set a step utility function that gives a very low utility after the deadline time instant. The value should be below any meaningful value of other successfully finished jobs, but greater than 0 as it is part of the presumption of our algorithm.

Also, we can consider the link stability in the ACO-based scheduling if we have a better model to predict it. For example, if we know at a future time point t the chance of a processor j to lose connection is $x(t, j)$, when calculating the "chance to select a trail" in Section. 2.4, we can multiply η_{ij} with $x(t, j)$. This will reduce the probability to choose an unstable node and increase the offloading success rate. As a further step, if we know the aforementioned chance, we can also schedule a job twice if its expected at finishing time t', the process it is scheduled to, say j, has a too low $x(t', j)$.

4.3 Conclusion

The work of this research resides mainly in two parts: first is the frameworks for generic applications and the workflows they work, second is the scheduling algorithms.

As we have discussed in the background, currently there is not a cloud computing framework for generic applications in current dynamic road environment. Existing work focuses on specific applications, like pictures and caching services, assumes a relatively static environment, like parking slots, and/or relies on infrastructures that are not ready in current state. Therefore the framework we proposed is to fill this gap. To do this, we divide functionalities in the architecture to modules, and weaken their correlations to allow more flexibility for software developers.

And then, the workflow of the proposed frameworks supports scheduling and offloading without centralized control. Therefore the frameworks can be applied in current road environment. Also with consideration to unstable V2V connections, the frameworks can perform well in the dynamic vehicular network, which is more common on roads.

Moreover, two algorithms are proposed to solve the scheduling problems. They focus on the optimization in decentralized network, and consider factors like host requirement and real-time applications. The ACO-based one, as we evaluated in AVE framework, has close outcome of a brute force search, with much less time consumed in computing the solution. And the one used in HVC framework is an online algorithm which is capable to find scheduling solution for jobs immediately upon their arrival.

Finally, we discuss some alternatives, while which is not our focus, in applying the work. Evaluation is done to test the potential of millimeter wave (mmWave) in vehicular cloud computing. Integration with centralized control is also discussed. And how to use the proposed ACO algorithm to solve other types of scheduling algorithm is briefly described.

4.4 Future work

4.4.1 Cloud computing in IoT (Internet of Things)

The work described here, while mainly designed for use in vehicular environment, can be extended to apply to other mobile cloud computing scenarios. The main idea behind the extension is that vehicular cloud computing is actually a special case of mobile cloud computing, with extra constraints like

movements speed and network coverage, which is insignificant in traditional mobile cloud computing scenarios, and also without limitations from battery life. With such difference considered, we can extend the vehicular cloud computing frameworks to generic mobile cloud computing scenarios.

A typical mobile cloud computing scenario is described in Fig. 4.1. Mobile devices, represented in the figure as "smartphone" icons, have direct ad-hoc connections with other devices. Some devices are also within the coverage of a cloudlet server or an access point. Just like vehicles, mobile devices generate jobs in applications. Some jobs can be processed by nearby devices or cloudlet servers, therefore they don't have to be offloaded to a remote cloud via cellular network.

Mobile devices have shorter transmission range than vehicles, but also with relatively slower movement. And in typical scenarios, obstacles are lesser problems for mobile devices comparing to communication range. But a factor to be considered is that unlike vehicles, mobile devices are usually constrained by battery life[89].

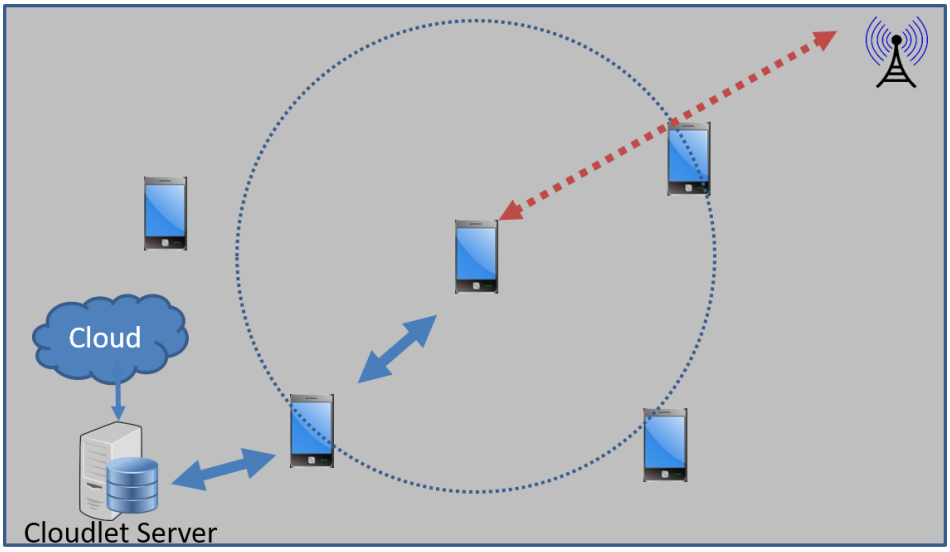


Figure 4.1: A mobile cloud computing scenario. Dashed circle denotes the ad-hoc communication range of the mobile device in the center. The mobile device in the center cannot reach cloudlet server, who has computation capability and may also connect to a centralized cloud, without relaying by the device in the bottom left corner. Most mobile devices have cellular network connection, which has a much larger communication range.

As an adaption to the new scenario, some modifications should be made on current proposal. An important one is to include power management into the architecture. For example, to extent AVE to mobile scenario, the framework should have ability to acquire battery level and power consumption of jobs, and add them in the utility function. The same goes for HVC extensions, where in addition to data usage, battery usage should be a factor in the objective function.

The workflows of the frameworks need adjustments too. Because the movement pattern is no longer restricted by road maps, it becomes harder to predict the links in mobile scenario. For relatively slower changing network, the predictions part can be removed. For other cases, better link duration model should be employed.

Also, scheduling algorithms can have some tweaks in parameters. For example, the ACO-based algorithm in AVE framework may reduce the default number of iterations, considering the hardware limitations on average mobile devices and the power consumption.

In addition to the scenarios in mobile environments, the extension of current work to other types of Internet of Things (IoT), like [90], is also viable. An example is for sensors with moving capability to acquire cloud computing services. When sensors have some data to be processed, the proposed cloud computing framework would provide them the solution for computational resources, either from remote cloud server or neighbor nodes.

4.4.2 Future of vehicular cloud computing

The smart vehicle industry is growing quickly, no one can exactly tell what will become the next hot spot. However, from observation of current trend, we can expect vehicles keep getting more and more intelligent, which means more applications will be developed and deployed to vehicular network. For example, we may have artificial intelligence assistant to help driver acquire information and control the vehicle, which requires large amount of computation with deep learning, voice and image recognition, etc.; gaming is not only heavily relying on computational capability, but also requires low latency.

Without such kinds of computationally intensive applications appearing, cloud computing can be expected to have a wider adoption in vehicular environments. With industry and government

investments, more infrastructures will be deployed, and more unified platforms will be available.

And last but not least, as vehicles becoming more and more powerful, there might be ideas emerging to exploit vehicle resources, like sensor, storage, or computation, to provide services to customers outside of the vehicular environment.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Prof. Yusheng Ji for the continuous support of my Ph.D. study and related research, for her patience, motivation, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study.

My sincere thanks also go to Dr. Zhi Liu, and Prof. Celimuge Wu, who helped me in the research and preparation of the papers. Without they precious support it would not be possible to conduct this research.

I thank committee members for giving valuable suggestions during my research, and in revising this thesis. And I thank all Sokendai staffs who assisted me in living and studying in Japan.

I also thank my fellow labmates in for the stimulating discussions, for the helpful suggestions to my research, and for all the fun we have had in the last five years. And my thanks go to my parents in China, who support me through all the years.

Bibliography

[1] Tim Triplett, Rob Santos, Sandra Rosenbloom, and Brian Tefft, "American Driving Survey 20142015," 2016.

[2] Chris Urmson and William Whittaker, "Self-driving cars and the Urban challenge," *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 66–68, 2008.

[3] J Rivington and M Swider, "Apple CarPlay: everything you need to know about iOS in the car," 2015.

[4] Michael Armbrust, Ion Stoica, Matei Zaharia, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, and Ariel Rabkin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50, apr 2010.

[5] Hye Sun Park, Min Woo Park, Kwang Hee Won, Kyong-Ho Kim, and Soon Ki Jung, "In-vehicle AR-HUD system to provide driving-safety information," *ETRI journal*, vol. 35, no. 6, pp. 1038–1047, 2013.

[6] Lin Che-Tsung, Lin Yu-Chen, Chen Long-Tai, and Wang Yuan-Fang, "Enhancing Vehicular Safety in Adverse Weather Using Computer Vision Analysis," in *VTC Fall, 2014 IEEE 80th*, 2014, pp. 1–7.

[7] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden, "CarTel: a distributed mobile sensor computing system," 2006.

[8] Xue Feng, Brigitte Richardson, Scott Amman, and James Glass, "On using heterogeneous data for vehicle-based speech recognition: A DNN-based approach," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. apr 2015, pp. 4385–4389, IEEE.

[9] G Russo, E Baccaglini, L Boulard, D Brevi, and R Scopigno, "Video processing for V2V communications: A case study with traffic lights and plate recognition," *2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pp. 144–148, 2015.

[10] John W Rittinghouse and James F Ransome, *Cloud Computing: Implementation, Management, and Security*, 2010.

[11] Guoqiang Hu, Wee-Peng Tay, and Yonggang Wen, "Cloud robotics: architecture, challenges and applications," *Network, IEEE*, vol. 26, no. 3, pp. 21–28, 2012.

[12] Liu Fangming, Shu Peng, Jin Hai, Ding Linjie, Yu Jie, Niu Di, and Li Bo, "Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 14–22, jun 2013.

[13] A Triviño-Cabrera, J Garcia-de-la Nava, E Casilari, and F J González-Cañete, "An Analytical Model to Estimate Path Duration in MANETs," in *Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, New York, NY, USA, 2006, MSWiM '06, pp. 183–186, ACM.

[14] T H Luan, Ling Xinhua, and Shen Xuemin, "MAC in Motion: Impact of Mobility on the MAC of Drive-Thru Internet," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 2, pp. 305–319, 2012.

[15] Mahmoud Abuelela and Stephan Olariu, "Taking VANET to the clouds," in *Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia - MoMM '10*, New York, New York, USA, 2010, p. 6, ACM Press.

[16] Ghada Moussa, Essam Radwan, and Khaled Hussain, "Augmented Reality Vehicle system: Left-turn maneuver study," *Transportation Research Part C: Emerging Technologies*, vol. 21, no. 1, pp. 1–16, apr 2012.

[17] S Tachi, M Inami, and Y Uema, "Augmented reality helps drivers see around blind spots," *IEEE Spectrum*, vol. 31, 2014.

[18] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young, "Mobile Edge Computing – A Key Technology Towards 5G," *ETSI White Paper*, vol. 11, pp. 1–16, 2015.

[19] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li, "Fog Computing: Platform and Applications," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. nov 2015, pp. 73–78, IEEE.

[20] Shaowei Yu, Qingling Liu, and Xiuhai Li, "Full velocity difference and acceleration model for a car-following theory," *Communications in Nonlinear Science and Numerical Simulation*, vol. 18, no. 5, pp. 1229–1234, 2013.

[21] Xueshi Hou, Yong Li, Min Chen, Di Wu, Depeng Jin, and Sheng Chen, "Vehicular Fog Computing: A Viewpoint of Vehicles As the Infrastructures," *IEEE Transactions on Vehicular Technology*, vol. 9545, no. 2013, pp. 1, 2016.

[22] Xiaoyan Kui, Yun Sun, Shigeng Zhang, and Yong Li, "Characterizing the Capability of Vehicular Fog Computing in Large-scale Urban Environment," *IEEE Transactions on Vehicular Technology*, 2016.

[23] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava, "A Survey of Computation Offloading for Mobile Systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, feb 2013.

[24] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems - EuroSys '11*. 2011, p. 301, ACM Press.

[25] Allan Bonnick, *Automotive computer controlled systems*, Routledge, 2007.

[26] I S O 11898:2015, "Road vehicles – Controller area network (CAN)," Standard, International Organization for Standardization, 2015.

[27] http://www.nvidia.com/en-us/self-driving-cars/drive-px/, "NVidia Driver PX," 2016.

[28] Paul Hansen, "BMW and Audi Want to Separate Vehicle Hardware from Software," apr 2017.

[29] Xiaopeng Fan, Jiannong Cao, and Haixia Mao, "A survey of mobile cloud computing," *ZTE Communications*, vol. 9, no. 1, pp. 4–8, 2011.

[30] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl, "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services - MobiSys '10*, New York, New York, USA, 2010, vol. 17, p. 49, ACM Press.

[31] Huang Dijiang, Zhang Xinwen, Kang Myong, and Luo Jim, "MobiCloud: Building Secure Cloud Framework for Mobile Computing and Communication," in *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, 2010, pp. 27–34.

[32] Lei Yang, Jiannong Cao, Shaojie Tang, Tao Li, and Alvin T.S. Chan, "A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing," in *2012 IEEE Fifth International Conference on Cloud Computing*. jun 2012, pp. 794–802, IEEE.

[33] Lei Yang, Jiannong Cao, Hui Cheng, and Yusheng Ji, "Multi-User Computation Partitioning for Latency Sensitive Mobile Cloud Applications," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2253–2266, aug 2015.

[34] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, oct 2016.

[35] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan, "Towards Wearable Cognitive Assistance," in *Proceedings of the 12th Annual Inter-*

national Conference on Mobile Systems, Applications, and Services, New York, NY, USA, 2014, MobiSys '14, pp. 68–81, ACM.

[36] S Arif, S Olariu, Wang Jin, Yan Gongjun, Yang Weiming, and I Khalil, "Datacenter at the Airport: Reasoning about Time-Dependent Parking Lot Occupancy," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 11, pp. 2067–2080, 2012.

[37] M Gerla, Weng Jui-Ting, and G Pau, "Pics-on-wheels: Photo surveillance in the vehicular cloud," in *Computing, Networking and Communications (ICNC), 2013 International Conference on*, 2013, pp. 1123–1127.

[38] Ma Meng, Huang Yu, Chu Chao-Hsien, and Wang Ping, "User-driven cloud transportation system for smart driving," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, 2012, pp. 658–665.

[39] Zhou Su, Yilong Hui, and Song Guo, "D2D-based content delivery with parked vehicles in vehicular social networks," *IEEE Wireless Communications*, vol. 23, no. 4, pp. 90–95, aug 2016.

[40] Zhou Su, Qichao Xu, Yilong Hui, Mi Wen, and Song Guo, "A Game Theoretic Approach to Parked Vehicle Assisted Content Delivery in Vehicular Ad Hoc Networks," *IEEE Transactions on Vehicular Technology*, vol. 9545, no. c, pp. 1, 2016.

[41] Meng Guo, Mostafa H. Ammar, and Ellen W. Zegura, "V3: A vehicle-to-vehicle live video streaming architecture," *Pervasive and Mobile Computing*, vol. 1, no. 4, pp. 404–424, 2005.

[42] Hossam S Hassanein, Sherin Abdelhamid, and Khalid Elgazzar, "A framework for vehicular cloud computing," *2015 International Conference on Connected Vehicles and Expo (ICCVE)*, pp. 238–239, 2015.

[43] Nianbo Liu, Ming Liu, Wei Lou, Guihai Chen, and Jiannong Cao, "PVA in VANETs: Stopped cars are not silent," in *2011 Proceedings IEEE INFOCOM*. apr 2011, pp. 431–435, IEEE.

[44] Sherin Abdelhamid, Hossam Hassanein, and Glen Takahara, "Vehicle as a resource (VaaR)," *IEEE Network*, vol. 29, no. 1, pp. 12–17, jan 2015.

[45] Ming Kai Jiau, Shih Chia Huang, Jenq Neng Hwang, and Athanasios V. Vasilakos, "Multimedia Services in Cloud-Based Vehicular Networks," *IEEE Intelligent Transportation Systems Magazine*, vol. 7, no. 3, pp. 62–79, 2015.

[46] Yi Ren, Fuqiang Liu, Zhi Liu, Chao Wang, and Yusheng Ji, "Power Control in D2D-Based Vehicular Communication Networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 12, pp. 5547–5562, 2015.

[47] Xinzhou Wu, Miucic R, Sichao Yang, Al-Stouhi S, Misener J, Sue Bai, and Chan Wai-hoi, "Cars Talk to Phones: A DSRC Based Vehicle-Pedestrian Safety System," in *Vehicular Technology Conference (VTC Fall), 2014 IEEE 80th*, 2014, pp. 1–7.

[48] Xu Chen, "Decentralized Computation Offloading Game For Mobile Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, apr 2014.

[49] Narayanan Sadagopan, Fan Bai, Bhaskar Krishnamachari, and Ahmed Helmy, "PATHS: analysis of PATH duration statistics and their impact on reactive MANET routing protocols," 2003.

[50] Seh Chun Ng, Wuxiong Zhang, Yu Zhang, Yang Yang, and Guoqiang Mao, "Analysis of access and connectivity probabilities in vehicular relay networks," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 1, pp. 140–150, 2011.

[51] Chisheng Zhang, Jiannong Cao, and Gang Yao, "CoDA: Connectivity-Oriented Data Dissemination Algorithm for Vehicular Internet Access Networks," *11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, pp. 186–193, 2015.

[52] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu, "The broadcast storm problem in a mobile ad hoc network," *Wirel. Netw.*, vol. 8, no. 2/3, pp. 153–167, 2002.

[53] Petra Schuurman and Gerhard J Woeginger, "A polynomial time approximation scheme for the two-stage multiprocessor flow shop problem," *Theoretical Computer Science*, vol. 237, no. 1-2, pp. 105–122, apr 2000.

[54] J A Hoogeveen, J K Lenstra, and B Veltman, "Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard," *European Journal of Operational Research*, vol. 89, no. 1, pp. 172–175, 1996.

[55] Charles E Perkins and Elizabeth M Royer, "Ad-hoc on-demand distance vector routing," *Proceedings - WMCSA'99: 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, 1999.

[56] V Namboodiri and Gao Lixin, "Prediction-Based Routing for Vehicular Ad Hoc Networks," *Vehicular Technology, IEEE Transactions on*, vol. 56, no. 4, pp. 2332–2345, 2007.

[57] Liu Ying, Liu Xiruo, W Trappe, and R Roy, "Distance-Aware Overlay Routing with AODV in Large Scale Ad Hoc Networks," in *Vehicular Technology Conference (VTC Fall), 2014 IEEE 80th*, 2014, pp. 1–5.

[58] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," in *Proceedings of the 5th European Conference on Computer Systems*, New York, NY, USA, 2010, EuroSys '10, pp. 265–278, ACM.

[59] S. Abrishami and M. Naghibzadeh, "Deadline-constrained workflow scheduling in software as a service Cloud," *Scientia Iranica*, vol. 19, no. 3, pp. 680–689, jun 2012.

[60] Guoxiang Zhang and Xingquan Zuo, "Deadline constrained task scheduling based on standard-PSO in a hybrid cloud," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7928 LNCS, pp. 200–209, 2013.

[61] Jianqiao Zhu, Ho-Leung Chan, and Tak-Wah Lam, "Non-clairvoyant Weighted Flow Time Scheduling on Different Multi-processor Models," *Theory of Computing Systems*, vol. 56, no. 1, pp. 82–95, jan 2015.

[62] Dirk G Cattrysse and Luk N Van Wassenhove, "A survey of algorithms for the generalized assignment problem," *European Journal of Operational Research*, vol. 60, no. 3, pp. 260–272, 1992.

[63] S Arora and M C Puri, "A variant of time minimizing assignment problem," *European Journal of Operational Research*, vol. 110, no. 2, pp. 314–325, 1998.

[64] Marco Dorigo and L M Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, apr 1997.

[65] Y.-C. Liang and A E Smith, "An Ant Colony Optimization Algorithm for the Redundancy Allocation Problem (RAP)," *IEEE Transactions on Reliability*, vol. 53, no. 3, pp. 417–423, sep 2004.

[66] David Martens, Manu De Backer, Raf Haesen, Student Member, Jan Vanthienen, Monique Snoeck, and Bart Baesens, "Classification With Ant Colony Optimization," *IEEE Transactions On Evolutionary Computation*, vol. 11, no. 5, pp. 651–665, 2007.

[67] Chandrasekharan Rajendran and Hans Ziegler, "Ant-colony algorithms for permutation flow-shop scheduling to minimize makespan/total flowtime of jobs," *European Journal of Operational Research*, vol. 155, no. 2, pp. 426–438, 2004.

[68] Varga Andras, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM'2001)*, 2001.

[69] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker, "Recent Development and Applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, 2012.

[70] C Sommer, R German, and F Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *Mobile Computing, IEEE Transactions on*, vol. 10, no. 1, pp. 3–15, 2011.

[71] L Codeca, R Frank, and T Engel, "Luxembourg SUMO Traffic (LuST) Scenario: 24 hours of mobility for vehicular networking research," 2015.

[72] Christoph Sommer, Stefan Joerer, and Falko Dressler, "On the applicability of Two-Ray path loss models for vehicular network simulation," in *2012 IEEE Vehicular Networking Conference (VNC)*. nov 2012, pp. 64–69, IEEE.

[73] C Sommer, D Eckhoff, R German, and F Dressler, "A computationally inexpensive empirical model of IEEE 802.11p radio shadowing in urban environments," in *Wireless On-Demand Network Systems and Services (WONS), 2011 Eighth International Conference on*, 2011, pp. 84–90.

[74] Paul Fuxjäger and Fabio Ricciato, "IEEE 802.11p Transmission Using GNURadio," in *6th Karlsruhe Workshop on Software Radios (WSR)*, 2010.

[75] A Mammeri, E H Khiari, and A Boukerche, "Road-Sign Text Recognition Architecture for Intelligent Transportation Systems," in *Vehicular Technology Conference (VTC Fall), 2014 IEEE 80th*, 2014, pp. 1–5.

[76] Mihail Sichitiu and Maria Kihl, "Inter-vehicle communication systems: a survey," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 2, pp. 88–105, 2008.

[77] Qi Chen, Daniel Jiang, and Luca Delgrossi, "IEEE 1609.4 DSRC multi-channel operations and its implications on vehicle safety communications," *2009 IEEE Vehicular Networking Conference, VNC 2009*, pp. 1–8, 2009.

[78] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Cáceres, and Nigel Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[79] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt, "Leveraging Cloudlets for Immersive Collaborative Applications," *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 30–38, 2013.

[80] Junil Choi, Vutha Va, Nuria González-Prelcic, Robert Daniels, Chandra R. Bhat, and Robert W. Heath, "Millimeter-Wave Vehicular Communication to Support Massive Automotive Sensing," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 160–167, 2016.

[81] Munsif Ali Jatoi and Nidal Kamel, "Introductory survey for wireless infrared communications," *The Journal of Engineering*, , no. October 2015, 2014.

[82] Sheng-hong Lin, Jin-yuan Wang, Xu Bao, and Yun Li, "Outage Performance Analysis for Outdoor Vehicular Visible Light Communications," *Wcsp2017*, pp. 0–4, 2017.

[83] Jonathan Wells, "Multigigabit wireless technology at 70 GHz, 80 GHz and 90 GHz," *RF Design*, pp. 50–58, 2006.

[84] "IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Am," *IEEE Std 802.11ad-2012 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae-2012 and IEEE Std 802.11aa-2012)*, pp. 1–628, dec 2012.

[85] Z Pi, J Choi, and R Heath, "Millimeter-wave gigabit broadband evolution toward 5G: fixed access and backhaul," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 138–144, apr 2016.

[86] J G Andrews, T Bai, M N Kulkarni, A Alkhateeb, A K Gupta, and R W Heath, "Modeling and Analyzing Millimeter Wave Cellular Systems," *IEEE Transactions on Communications*, vol. 65, no. 1, pp. 403–430, jan 2017.

[87] Celimuge Wu, Tsutomu Yoshinaga, and Yusheng Ji, "Cooperative Content Delivery in Vehicular Networks with Integration of Sub-6 GHz and mmWave," *IEEE GLOBECOM Workshop on 5G Networks Using Unlicensed Spectrum*, 2017.

[88] Qitu Hu, Celimuge Wu, Xiaobing Zhao, Xianfu Chen, Yusheng Ji, and Tsutomu Yoshinaga, "Vehicular Multi-access Edge Computing with licensed Sub-6 GHz, IEEE 802.11p and mmWave," *IEEE Access*, 2017.

[89] M Ali, J M Zain, M F Zolkipli, and G Badshah, "Mobile cloud computing and mobile battery augmentation techniques: A survey," in *2014 IEEE Student Conference on Research and Development*, dec 2014, pp. 1–6.

[90] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, feb 2014.