RESOURCE ASSIGNMENT ALGORITHMS FOR VEHICULAR CLOUDS

MAHMUDUDN NABI
Bachelor of Science, Islamic University of Technology, 2011

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

RESOURCE ASSIGNMENT ALGORITHMS FOR VEHICULAR CLOUDS

MAHMUDUDN NABI

Date of Defense: December 16, 2016

Dr. Robert Benkoczi Supervisor	Associate Professor	Ph.D.
Dr. Shahadat Hossain Committee Member	Professor	Ph.D.
Dr. John Zhang Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

To my parents.

Abstract

In this thesis, we study the task scheduling problem in vehicular clouds. It falls in the category of unrelated parallel machine scheduling problems. Resource assignment in vehicular clouds must deal with the transient nature of the cloud resources and a relaxed definition of non-preemptive tasks. Despite a rich literature in machine scheduling and grid computing, the resource assignment problem in vehicular clouds has not been examined yet. We show that even the problem of finding a minimum cost schedule for a single task over unrelated machines is NP-hard. We then provide a fully polynomial time approximation scheme and a greedy approximation for scheduling a single task. We extend these algorithms to the case of scheduling n tasks. We validate our algorithms through extensive simulations that use synthetically generated data as well as real data extracted from vehicle mobility and grid computing workload traces. Our contributions are, to the best of our knowledge, the first quantitative analysis of the computational power of vehicular clouds.

Acknowledgments

I owe my deepest gratitude to my supervisor Dr. Robert Benkoczi who has been providing endless support over the duration of this program. He has guided me into the world of optimization. Without his continuous enthusiasm, encouragement and optimism, this research endeavor would hardly have been completed.

I also take great pleasure in acknowledging the support of one of my MSc. supervisory committee members Dr. Shahadat Hossain, who has been constantly providing me invaluable insights, guidelines, and inspiration. I would like to thank my other committee member Dr. John Zhang for his valuable feedbacks which were essential towards the end of this work. In addition, I would also like to take the opportunity to thank Dr. Daya Gaur for his constructive suggestions.

I was very lucky to have the "Dean's scholarship and Tuition Award" from the School of Graduate Studies and would like to give them big thanks. I am deeply grateful to my supervisor for the financial assistance he has provided to me as his Research Assistant.

I am thankful to all the members of the **Optimization Research Group** of the University of Lethbridge for their continuous support and encouragement throughout my thesis period. Especially, I should mention Ram, Mark, and Umair for their help whenever needed. I also express my gratitude to my friends *Kawsar, Lazima, Marzia, Imtiaz, Jeeshan, Mamun, Jubair, Masroor, Moin, Tafseer* and many other friends from my Bangladeshi Community, who have been the source of my motivation and moral support during my stay in Canada.

I am indebted to my parents and my sister, who have been supporting me for my whole life. Their sacrifices and prayers are the main driving forces that enabled me achieving my goals. Thank you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past few years, Intelligent Transportation System (ITS) has received much attention due to the wide scope of benefits offered by the vehicles. The Vehicular Ad-hoc Network (VANET), an essential part of ITS, has emerged to connect vehicles on roads [31]. Connected vehicles in a VANET are known as **smart vehicles** and are considered to be the next frontiers for the Internet of Things (IoT) [12] [42].

Cloud computing is an emerging technology that provides cost-effective services to users by allocating resources (e.g.-storage, CPU, etc) on demand for processing tasks. Cisco introduced the concept of *fog computing* to deal with Big Data analytics and applications in the IoT [13, 14]. Fog or *edge computing* attempts to improve the performance of systems through computations that are closer to the customers requesting the service rather than being located in far-off data centers [18, 19, 20]. The vehicular cloud (VC) is a particular implementation of fog computing.

The concept of "vehicular clouds" (VCs) was introduced by Olariu *et al.* [45] about the same time when Cisco was promoting the notion of fog computing. Their motivation was to take the vehicular networks to the clouds.

In vehicular clouds, smart vehicles are equipped with components that have sensing, computing, and communication capabilities which can be harvested for data storage, computing, infotainment, and sensing services [8, 43, 53]. The main motivation for coming up with vehicular cloud paradigm was to utilize the underutilized vehicular resources in computing tasks by allocating them to the authorized users instead of limiting them to ITS

applications [43] [53]. As a result, the benefit of vehicular resource utilization would be maximized when a pool of vehicular resources are dynamically coordinated for a computing task compilation. Sherin et al. [8] introduced the concept of *Vehicle as a Resource* (VaaR) , where it is anticipated that a smart vehicle can be considered as a mobile resource provider and will be a key enabler for the revolution of the Internet of Things technology.

However, the vehicles are not owned by the service providers and they are a transient computation resource. Coordinating such computing resources requires solving scheduling problems with specific constraints. Scheduling plays a crucial role in terms of proper resource utilization. It is concerned with the allocation of limited resources to tasks over time. In vehicular clouds, incentives are offered to the vehicle owners to encourage renting their resources to the service providers, thus minimizing renting cost is one of the goals of the scheduler.

In this thesis, we focus on the task scheduling problem in vehicular clouds. There is an impressive amount of literature on scheduling tasks in domains such as machine scheduling [39], distributed, and grid computing [21]. Most popular objectives for the task scheduling problems are focused on quality of service parameters, such as makespan or lateness subject to the constraints like arbitrary precedence constraints, preemption, deadline etc. In the vast majority of the problems, the computation resources are always available. This assumption works for some scenarios but cannot be applied in many real-life applications. Because most of the real-life situations are dynamic in nature which means the input information is updated frequently. This assumption has been relaxed in machine scheduling problems with availabilities [50] in which the processors are available to execute jobs only during certain moments in time. The objectives most studied in problems of scheduling with availabilities are minimizing maximum completion time, minimizing total completion time, and minimizing maximum lateness [15]. Difficult instances of the scheduling problem with availabilities are usually for problems where preemption (see Section 2.6) is not allowed.

In contrast, VCs do not have any control over the availability of the computational and storage resources due to vehicle mobility. It makes the scheduling environment dynamic and the task scheduling more challenging as the system must be able to respond in real time to events triggered by resources becoming unavailable as vehicles move away from the area that defines the VC. Hence, tasks scheduled on a particular resource in a vehicular cloud can be paused, transferred to another resource, and resumed on that resource. This is needed for example, when a particular vehicle leaves the cloud and all of the jobs assigned to it need to be transferred to other vehicles. In addition, for VCs with no infrastructure support to store the state of running processes, a task cannot be paused and resumed at a later time because the resource (or the vehicle) storing its state may leave the cloud.

Therefore, we are interested in a task scheduling problem with availabilities subject to a relaxed notion of non-preemption: once a task is started on a resource, it must be executed to completion without interruption, except when execution is transferred to another resource and is resumed immediately. We call this constraint *VC-preemption*.

Scheduling tasks in a vehicular system has two optimization goals, minimizing makespan and minimizing cost. The makespan represents the total duration that a task takes from the moment it begins execution until it completes execution. On the other hand, each vehicle in VCs is associated with a rental cost based on its resource capabilities and features. Hence the cost objective aims at minimizing the total cost paid for the rented vehicles. In this thesis, we have focused on the second objective. Therefore, the objective of our problem is to minimize the cost of the schedule subject to the VC-preemption constraint with a common task deadline.

Research on scheduling problems in vehicular clouds subject to availability constraint has not been dealt with adequately in the literature and designing efficient algorithms for this problem is of great interest.

1.1 Motivation

The main motivation of this thesis is to utilize the untapped on-board computing resources of modern smart vehicles to perform different computing tasks. Present day vehicles are equipped with an in-vehicle processing unit which is as powerful as personal computers. Most of these vehicles are parked in a parking lot (or garage etc.) or spend time in road intersections (or traffic jams/driveway/around a location of interest etc.) while their vehicular resources remain underutilized. These resources can be rented out from the vehicle owners with appropriate incentives and can be utilized to carry out computing tasks offloaded by the service providers for providing various services (e.g.- navigation, entertainment, traffic information, weather information etc) to the users.

The automotive industry is doing continuous research to bring a revolution of smart vehicles on the road. Recently, Tesla Motors has released an autopilot software that enables their Model S sedans with autonomous driving capabilities [1]. Ford is using a cloud computing and in-car software to provide new services to the car owners and also working on autonomous driving cars [3]. Futhermore, NVIDIA released a connected car technology that has their Tegra X1 Visual Computing Module which integrates audio, video, and image processors [4]. Audi A8 uses NVIDIA processor to power its 3D navigation system display. Recent Volkswagen, Honda, and Mercedes cars are also equipped with processors, GPS, video camera, sensors etc. [2]. The automotive companies have already brought the dream of smart cars into reality and continuously working to make the future vehicles as the most powerful computers by packing the power of a supercomputer inside a car. Therefore, these smart vehicles are beyond transportation machines.

Nowadays the cloud computing paradigm has emerged with its advanced capabilities that have encouraged users to move their services such as computations, IT services etc. into the cloud infrastructure. Concurrently, the smart vehicles are the machines with computing power. The vehicular cloud computing technology can provide greater benefits to the users by forming a vehicular cloud by combining the resources from multiple vehicles.

The available processing powers of the vehicles in the VC can be utilized as computing engines to carry out different computing tasks offloaded to them by the users. Thus the VC can be used as the traditional cloud system. However, to be able to utilize the in-vehicle computing resources, the vehicles need to be powered while parked if they are to participate in the cloud.

One challenge for the user (or service provider) is the assignment of the computational tasks among the resources based on their availability span and within a limited budget. It is, therefore, necessary to explore a cost-effective dynamic resource allocation strategy for the VC system with the interval availability constraint.

1.2 Contributions

To the best of our knowledge, no previous study of a machine scheduling problem with the VC-preemption constraint has been carried out in the literature we surveyed. The following are our contributions in this thesis.

- We show that the single task scheduling problem on a set of unrelated machines to minimize scheduling cost subject to a task deadline and VC-preemption is NP-hard, by exploiting the connection of the problem with knapsack cover (KC) [37, 24].

- We provide a fully polynomial time approximation scheme (FPTAS) (see Section 2.5.1) for the single task scheduling problem by extending the idea used in an FPTAS for knapsack cover [34] to our problem with VC-preemption with a performance ratio of $(1+\varepsilon)$.

- We describe a natural greedy algorithm for the single task scheduling problem on unrelated machines. We note that our greedy algorithm is trivially optimal for the version with identical machines.

- We provide a natural greedy algorithm to schedule n tasks on unrelated machines with VC-preemption that schedules one task at a time by repeatedly calling a single

task scheduling procedure for vehicular cloud.

- We give a simple and powerful lower bound on the cost of the optimal solution for scheduling n tasks by solving a knapsack cover problem fractionally [25]. We use this bound to calculate approximation ratios of our algorithm for the n tasks scheduling problem on an extensive set of problem instances.

- We perform a comprehensive empirical evaluation of our task scheduling algorithm for the n tasks scheduling problem with both the greedy and PTAS (see Section 2.5.1) procedures for solving the single task scheduling problem, using both synthetically generated data and real data extracted from vehicle mobility traces and grid workload traces. We note that the scheduling problems studied here are off-line, i.e. the set of tasks and the availability of the resources are known at the start of the simulation. We observed that the result of the greedy procedure for scheduling n tasks is very close to the lower bound proposed. On the synthetic problem instances, the average gap between the lower bound of the schedule cost and the solution returned was less than 2.5%. Moreover, on the real data instances, the approximation ratio was larger but not larger than 25%. These findings are pivotal for the evaluation of a vehicular cloud task scheduler in an on-line setting where tasks and resource availabilities become known with time. A competitive analysis of the scheduling algorithm with real data involves instances with millions of tasks and the bounds on the cost of the optimal solution in the off-line setting must be powerful and extremely fast to compute.

- We provide the first, as far as we know, quantitative evidence on the processing capability of a vehicular cloud using real life grid workload traces and considering the transient nature of the cloud resources and the specifics of task VC-preemption constraints. We observe that more than 92% on average of the grid processes were scheduled successfully on the vehicular cloud in the most constrained of the instances, and, when given some slack, a vehicular cloud can serve all but a handful of the most com-

pute intensive jobs which need to be offloaded to the classical cloud.

1.3 Thesis organization

The rest of this thesis is organized in the following order. Chapter 2 provides the background information related to the vehicular paradigm, smart vehicles, cloud computing, vehicular cloud and task allocation and scheduling. Chapter 3 presents our main contribution. The problem definition, proposed algorithms for the resource assignment problem in vehicular clouds are described in this chapter. In Chapter 4, we presented the implementation details and the experimental results of our algorithms. Finally, in Chapter 5, the concluding remarks and the future research directions are mentioned.

Chapter 2

Background

Vehicular Cloud (VC) is a hybrid technology that has emerged by the integration of vehicular networks and cloud computing paradigms. The research areas that are directly related to this VC system are vehicular ad-hoc networks, smart vehicles, cloud computing, machine scheduling. The main problem that is focused in this thesis is the resource assignment problem in vehicular cloud paradigm which falls in the category of unrelated parallel machine scheduling problem. In this chapter, at first, we discuss the research areas related to the vehicular cloud. Then we discuss the concepts of the run time complexity, the approximation algorithms and scheduling problems. Finally, we describe some previous works which are related to solving the task allocation problem in vehicular clouds.

2.1 Vehicular Networks

For the past few years, the research efforts on Vehicular Ad-hoc Networks (VANETs) have increased due to an attractive range of applications such as traffic safety and control [10, 31, 35]. VANET provides the state-of-the-art services for traffic management and transportation by means of vehicular communication where safety, navigation, road condition etc. information are exchanged between the vehicles on the road. A VANET is a kind of the wireless multi-hop networks that depend on multi-hop communication over intermediate nodes working as relays to connect a source node to a destination node. The emerging capability of connecting vehicles on roads has led VANET to converge with ITS and enhance the transportation efficiency and safety. Connected vehicles in a VANET are

known as “smart vehicles”.

2.1.1 Smart Vehicles

A vehicle is called “smart” when it is equipped with components such as GPS, different types of sensors for monitoring the surroundings of the vehicle for road safety etc. The main component of a smart vehicle is the on-board unit (OBU), which is as powerful as a personal computer and also known as the in-vehicle PC. The OBUs of smart vehicles have processing and storage capabilities [51]. An OBU works as the interface that provides the driver with information/alerts about events that are either detected by the in-vehicle sensors or received through the communication module. It is also the means of receiving input from the driver when needed. In addition, it has an on-board wireless communication unit to communicate with nearby vehicles and road side units (RSUs).

A smart vehicle with these components available on-board has become more sophisticated and capable of offering more diverse services such as sensing, storage, computing, relaying, infotainment, and localization [8]. Infotainment is a service that delivers a combination of information and entertainment. Typical contents of infotainment include managing and playing audio content, utilizing navigation for driving, making phone calls, accessing traffic information through the Internet etc. The plethora of vehicular resources has made smart vehicles as the key enablers in service provisioning, compared to other mobile resource providers, such as smart-phones.

An example of the in-vehicle components of smart vehicle is shown in Fig. 2.1 which was provided in the thesis of Sherin [6].

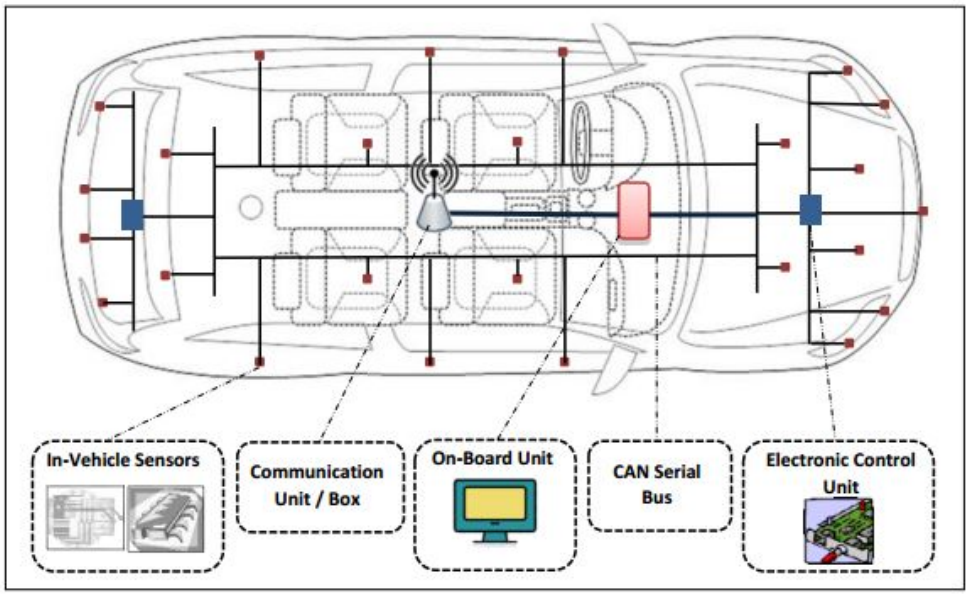


Figure 2.1: In-vehicle components of a smart vehicle.

2.2 Cloud Computing (CC)

Cloud computing is a state-of-the-art technology that offers a wider scope of services to the users over the Internet. It makes various computational resources available to the users in a cost effective manner. The formal definition of cloud computing is given by Mell and Grance from the National Institute of Standards and Technology (NIST) as follows [41]:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

The cloud computing service providers offer three types of services namely- infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS).

IaaS In the case of IaaS, fundamental computing resources such as servers, network devices, and storage disks are made available to the organizations as services on a need-to basis. Many IaaS providers (e.g.- Amazon AWS, Windows Azure, Google Computer En-

gine, Rackspace, IBM SmartCloud etc.) with the help of virtual machines (VMs) offer their computational resources in a "pay as you go" manner.

PaaS PaaS provides a platform for application development to the software developers. More specifically, it offers a range of software development tools for developing mobile and web applications to be deployed in the cloud. PaaS providers such as Engine Yard, Google AppEngine, Red Hat OpenShift, AppFog etc. offers a variety of programming languages and storage facilities to the developers.

SaaS SaaS allows customers to use the applications hosted by the SaaS providers in a cloud. In general, with SaaS users only need to think about how they will use the applications and do not need to think about how it is maintained. The service providers are responsible for the maintenance of their services as well as managing the underlying infrastructures. Some renowned SaaS providers are Oracle, Microsoft, Salesforce, Intuit etc.

2.2.1 Pros and Cons

Following are the key advantages of cloud computing.

- It reduces the investment cost by providing the hardware and software support to the users.
- It offers computing power and IT services on demand with increased scalability.
- It is capable of supporting new and innovative applications that are not supported in a traditional IT environment without investing large capital for the computational infrastructure.

However, along with the above advantages, cloud computing suffers from the security and privacy issues as the customers have little control on the physical devices that execute their applications and store their data due to virtualization. Thus concerns regarding privacy

and compliance with local laws have been raised. As a result, the US Federal Government, European Union Agency for Network and Information Security (ANISA), Canadian government agencies etc. have taken appropriate cloud adoption strategies to evaluate and monitor the security and suitability of cloud solutions [23, 22, 48]. Moreover, along with the security concerns cloud users have limited control over the cloud functionalities and the infrastructures which are entirely owned and managed by the service providers. Cloud computing is fully dependent on Internet connection. That means without Internet users will not be able to access the cloud services which is another disadvantage of cloud computing.

2.3 Fog Computing

Fog computing also is known as *edge computing* or *fogging* that extends the cloud computing paradigm to the edge of the networks [14]. It brings the advantages and power of the cloud to where data is created [18, 19, 20]. Fog computing facilitates the operation of compute, storage and networking services between the data source and the cloud. The term *fog* refers to a cloud which is closer to the ground, just as fog concentrates on the edge of the network. The goal of fogging is to improve efficiency and reduce the amount of data transported to the cloud for processing, analysis and storage. It is a perfect platform that has been evolved to support the Internet of Things (IoT) applications such as connected vehicles, smart grid, smart cities etc. Cisco introduced the concept of fog computing to enable systemetic, secure and network-integrated computing and storage services between the cloud and end-users [13, 14].

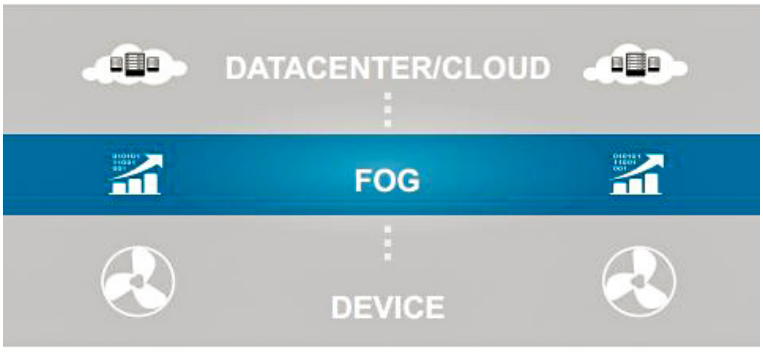


Figure 2.2: Relation between cloud, fog and end-device.

Figure 2.2 is presented in the Cisco white paper [5] that shows how the fog extends the cloud closer to the devices that produce data. These devices are called the fog nodes. Any device with computing, storage, and network connectivity can be a fog node. They can be deployed anywhere with a network connection: on a factory floor, on top of a power pole, alongside a railway track, in a vehicle, or on an oil rig. The edge devices and sensors those produce data, do not have the compute or storage resources to perform advanced analytics. Though cloud servers have the power to do these but they are often too far away to process the data and respond in a timely manner. Therefore, analyzing data close to where they are collected minimizes latency. It offloads gigabytes of network traffic from the core network. And it keeps sensitive data inside the network [5]. The major differences between the cloud and the fog are the proximity to end-users, geographical locations of the nodes and support for mobility. Connected vehicles of VANET is a particular implementation of fog computing infrastructure. It can support services like infotainment, traffic control, safety, mobility, and location awareness etc [14].

2.4 Vehicular Clouds (VC)

The on-board computational resources of smart vehicles has motivated the researchers to exploit these underutilized resources for providing a wider scope of services to the users beyond just transportation. As a result, the “vehicular cloud” (VC) paradigm has emerged.

In this section, we define the vehicular cloud platform along with its different architectures and potential applications of this paradigm.

2.4.1 Definition

Nowadays many emerging applications require high computational power as well as large storage facilities. Vehicular resources from multiple vehicles can be shared or combined together to fulfill the requirements of these applications. This has become the main motivation for the study of cloud-based vehicular networks with the futuristic vision of "taking vehicles to the cloud". In essence, the vehicular cloud can be defined as follows [44]:

"*A vehicular cloud is a collection of smart vehicles whose vehicular resources are combined dynamically and allocated to the authorized users to perform a computing task on demand.*"

VCs can provide the on-demand solutions for services like entertainment, road safety, weather report analysis, real-time traffic information etc. Like the conventional cloud systems in VCs the computing resources are rented out to authorized users based on a rental model. Besides the similarities, there are several advantages of a VC system compared to the fixed cloud computing. Areas with limited Internet access as well as with inadequate cloud facilities can be served by a VC because of the vehicular mobility. Additionally, during the events of unexpected occurrences (e.g., the natural disasters or any emergency situations) where the basic communication infrastructures may be broken down, VCs can be formed and served as a communication medium. A detailed study on vehicular cloud computing is presented in [53].

2.4.2 Architectuers

The architecture of a vehicular cloud system can be categorized into two different types: i) centralized and ii) autonomous. These two types are discussed below with example scenarios for each of them.

Centralized Vehicular Clouds

A centralized VC architecture consists of a central cloud controller interacting with node controllers. The cloud controller is the central entity of this VC that manages the computing resources. It performs the resource discovery, task allocation, and the data exchange operations between the participating vehicles with the help of three major components - a broker, a resource manager, and a task scheduler. This architecture follows a client / server model where the clients request for accessing the computing resources to the cloud controller. The broker deals with the client resource request on behalf of the VC. The resource manager is mainly responsible for the resource availability check and resource allocation to the tasks (i.e. - creating access schedule to the resources) in cooperation with the task scheduler based on the resource availability span.

Each vehicle interested in participating in a VC interacts with the cloud controller with the help of an on board interface called node controller. It works as a local resource manager for that vehicle.

If a VC is large in size (defined by the number of participating vehicles), it can be divided into clusters with each cluster being managed by a cluster controller. Each cluster controller works between the cloud controller and the node controller, and monitors and manages the resources of its own cluster that reduces the management load of the cloud controller.

An example of centralized VC can be the VC formed at a parking lot (or parking garage/driveway etc.). Everyday, many vehicles are parked in a parking lot for several hours with each having different (or same) arrival and departure times, as vehicles do not stay in the parking lot forever. These vehicles are parked idle with ample unexploited computing resources where the resource capabilities of each vehicle are different (or same). Vehicles with these amount of untapped resources are the perfect candidates for nodes for a cloud system. These resources can be rented out from the vehicle owners with appropriate incentives based on their capabilities and can be utilized to carry out computing tasks offloaded

to it instead of renting a computing infrastructure. Both the parties benefit economically here. Moreover, the parked vehicles need to be connected to a power supply and a data port offering Internet access. Without any power supply, the parked vehicles cannot have their on-board computers and resources on all the time waiting for task assignments. Figure 2.3(a) depicts an example of a centralized VC formed at a parking lot which is provided in [7].

Autonomous Vehicular Clouds

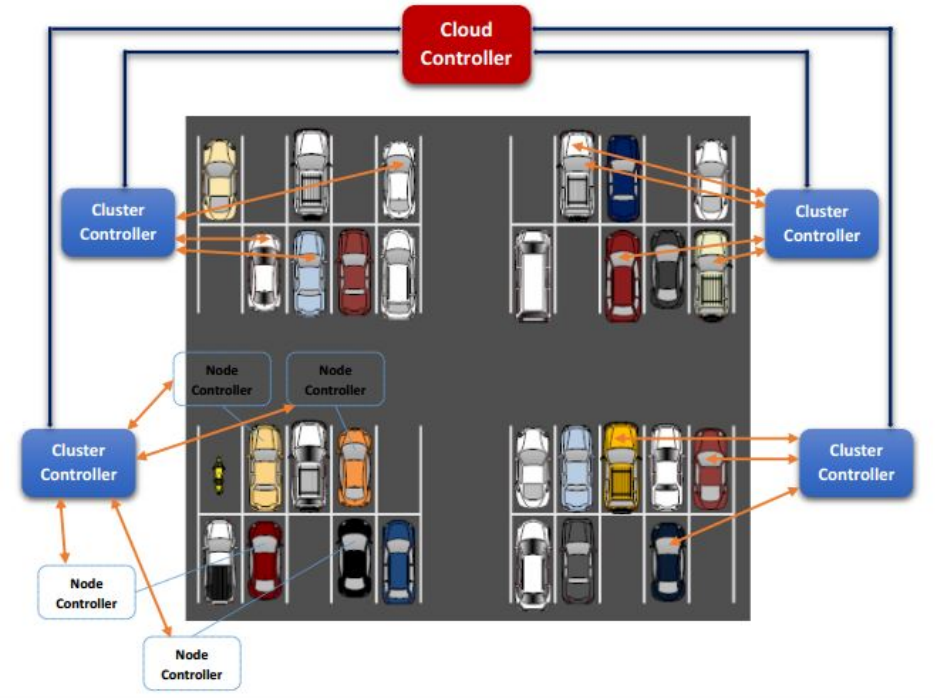
An autonomous VC can be formed temporarily in a self-organizing fashion without the help of a central entity (i.e. cloud controller). For example, in some scenarios like- traffic jam, an intersection of a road etc. a VC can be formed to help the city traffic authorities to alert and re-route vehicles, rescheduling traffic lights to mitigate the congestion. In such a case, computing resources from the vehicles in the vicinity of the event are self-organized to handle the required task.

An autonomous VC has short life span with low computing requirements compared to the relatively long-lasting centralized VC. Thus it does not involve a central cloud controller along with the broker component like the well-planned centralized VC. It requires limited resource management and task scheduling functionalities that are managed by a VC coordinator (e.g., a vehicle elected from the participating vehicles forming the VC). Similar to the centralized VCs, vehicles can be grouped into clusters and managed by a cluster controller. In autonomous VCs, the cluster controllers create links between the node controllers and the VC coordinator, when needed.

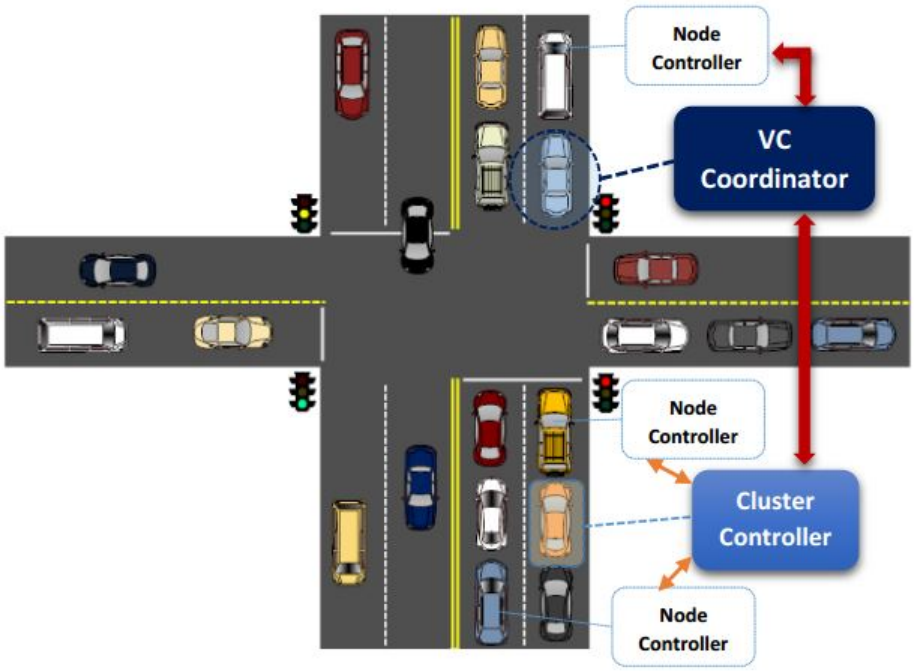
Figure 2.3(b) depicts a the architecture of an autonomous VC which is given in [7].

2.4.3 Services of VC

The types of services a VC can provide are described as follows.



(a) Architecture of a centralized VC.



(b) Architecture of an autonomous VC.

Figure 2.3: The different architectures of vehicular clouds [7].

Processing as a Service (PRaaS)

With on board unit (OBU) a smart vehicle has the computing capabilities of a personal computer which can be utilized to perform processing tasks. While vehicles spend several hours in the parking lot (or garage or driveway) their idle processing resources can be used by the authorized users by renting them from the vehicle owners. Aggregated resources from multiple vehicles can be utilized to handle tasks with high processing requirements that exceed a single vehicles capabilities. Efficient task scheduling mechanisms are needed to handle such scenario.

Storage as a Service (STaaS)

Smart vehicles are anticipated to have plenty of on-board storage capability (e.g.- Terabytes of storage). Vehicles with this capability can provide storage as a service [11]. Vehicles can share their storage resources with others (i.e. service providers) in need of storage with appropriate security measurement. For example, storage resources from the vehicles parked in an airport parking lot can be used as a data center in return of free parking facility. Due to the mobility, vehicles can support their use as data mules carrying data between a pair of nodes in need of data delivery which is an advantage over the traditional cloud system.

Network as a Service (NaaS)

Some vehicles can connect to the Internet while moving and can offer internet access to other vehicles in a 'Drive-by Internet Access' model like mobile hotspots. However, many vehicles do not have this capability of Internet connection. Vehicles having this capability and having the interest to share their resources advertise this offer to the vehicles interested in renting it.

Information as a Service (INaaS)

Smart vehicles are considered as the storehouse of information collected by their advanced on-board equipments (e.g.- sensors, GPS etc). Vehicles in a VC can store information like - traffic information, weather or road conditions, traffic jam warnings, etc. obtained from other vehicles or road side units (RSUs) by utilizing their sensors. Depending on the information type the service provider who rented the vehicular resources from the owner can resolve queries generated from the other moving vehicles or the vehicles outside of the VC.

2.4.4 Applications of VC

With the above discussed service scopes, there are several applications where VC paradigm can be utilized. As mentioned above our thesis mainly focuses on the first service type, so we will briefly discuss some applications of this service type at first. Later we will outline some applications of the rest of the services.

Computing Engines at a Parking Lot

Resources from the vehicles parked idle in a parking lot can be utilized as computing engines that can carry out computing tasks offloaded to them by the user. These combined vehicular resources work as a computing infrastructure for the user renting the resources where both the user and the vehicle owner get the benefit. This application can be considered as an example of PRaaS and/or STaaS services.

Dynamic Traffic Light Managemant

An autonomous VC of vehicles formed at an intersection can help in controlling the city traffic by dynamically rescheduling the traffic lights of the city. On board computational resources of the vehicles are used to run complex simulations designed to control the congestion of a road. This application also falls in the category of PRaaS service type.

More Applications There are many other potential application of VC cloud paradigm such as - Vehicular Public Sensing, airport parking lot as a data center, shopping mall data center, managing evacuation during natural disasters, road safety, weather condition analysis, managing parking facilities, autonomous congestion alleviation etc.

2.5 Computational Complexity

The problems that we consider appear in two versions: (i) decision version and (ii) optimization version. In decision problems, the solution (or the answer) is simply "yes" or "no". On the other hand, in the optimization problems, each feasible solution has an associated value, and the feasible solution with the best value is chosen as the final solution. In other words, the solution for any optimization problem is given as minimum (or maximum) of an objective.

We introduce some important definitions of complexity theory from the book [16].

***P*:** The class P is the set of all decision problems which can be solved in polynomial time. More specifically, these are the problems which can be solved in time $O(n^k)$ for some constant k, where n is the input size to the problem.

***NP*:** The class NP is the set of problems which can be "verified" in polynomial time. That means if we were given a "certificate" of a solution, then we could verify that the certificate is correct in polynomial time in the size of the input to the problem. Any problem in P is also in NP. Thus $P \subseteq NP$.

Polynomial-time Reduction: Suppose, we are given two problems P_1 and P_2. If there is a polynomial time function f such that, I is a "Yes" instance of P_1 if and only if $f(I)$ is a "Yes" instance of P_2, we can say that, P_1 is polynomially reducible to P_2. We represent it as $P_1 \leq_p P_2$. In other words, each instance of P_1 can be transformed to an instance of P_2 in polynomial time in the size of the instance.

***NP*-complete:** A problem is *NP*-complete if it is in *NP* and is as "hard" as any problem in *NP*. Formally, a problem P_2 that is in *NP* is also in *NP*-complete if and only if every other problem P_1 in *NP* can be reduced to P_2 in polynomial time. It means that we can solve P_1 quickly if we know how to solve P_2 quickly. In other words, we can say P_2 is *NP*-complete if-

1. $P_2 \in NP$, and
2. $P_1 \leq_p P_2$ for every $P_1 \in NP$.

NP-completeness does not apply directly to the optimization problems. It applies to the decision problems.

***NP*-hard:** A problem X is *NP*-hard if all other *NP* problems can be reduced to X, but X not necessarily belongs to *NP*. This problem is as hard as any other problem in *NP*.

2.5.1 Approximation Algorithms

For any *NP*-hard problem, it is unlikely to find an exact optimal solution within polynomial time in the size of an instance I. Therefore, *approximation algorithms* are proposed for this kind of problems which give near-optimal solutions in polynomial time. The formal definition of approximation algorithm is given as below.

"*Let X be a minimization (respectively, maximization) problem. Let* $\varepsilon > 0$*, and set* $\rho = 1+\varepsilon$ *(respectively,* $\rho = 1-\varepsilon$*). An algorithm A is called a* ρ*-approximation algorithm for problem X, if for all instances I of X it delivers a feasible solution with objective value* $A(I)$ *such that* $|A(I) - OPT(I)| \leq \varepsilon.OPT(I)$*. In this case, the value* ρ *is called the performance guarantee or the worst case ratio of the approximation algorithm A.*"

Here, $OPT(I)$ is the optimal objective value for instance I.

Approximation Scheme An approximation scheme for an optimization problem is an approximation algorithm that takes both an instance of the problem and a value $\varepsilon > 0$ as

input, such that the scheme is a $(1+\varepsilon)$-approximation algorithm, for any fixed ε. There are two types of approximation schemes namely polynomial time approximation scheme (PTAS) and fully polynomial time approximation scheme (FPTAS).

An approximation scheme is a polynomial-time approximation scheme (PTAS) if its time complexity is polynomial in the input size, for any fixed $\varepsilon > 0$. The running time of a PTAS increase very rapidly as ε decreases.

An approximation scheme is a fully polynomial-time approximation scheme (FPTAS) if it is an approximation scheme and its running time is polynomial in the input size and also polynomial in $1/\varepsilon$.

Approximation Ratio The ratio between the solution obtained from an approximation algorithm and the optimal solution for the same problem instance is known as the approximation ratio of that algorithm. Let, $A(I)$ be the solution returned by algorithm A on instance I and $OPT(I)$ is the optimal solution on the same instance, then the approximation ratio (AR) of algorithm A on instance I is,

$$AR = \frac{A(I)}{OPT(I)}.$$

2.6 Scheduling Problem

A scheduling process is concerned with the allocation of the limited resources (e.g. processors, machines etc.) to a group of tasks in order to optimize a particular objective function.

A scheduling problem can be specified in terms of the three-field notation scheme, denoted by $\alpha|\beta|\gamma$. The meaning of each of these fields can be adapted based on the problem type. The first two fields are generally used to represent the task and machine environments (in any order) and the third field can be used to represent the constraints or to describe the optimality criteria of the problem. The machine environment can have either a single machine or parallel machines. Parallel machines can be categorized into three types:

(i) *Identical* The machines are identical and any task requires same amount of time for processing on any machine.

(ii) *Uniform* Each machine i has a speed s_i and any task j with processing requirement p_j requires p_j/s_i time to process on machine i.

(iii) *Unrelated* Machines have different capabilities. Speed of machine i on task j s_{ij}, depends on both machine and the task. Task j requires p_j/s_{ij} processing time on machine i.

Moreover, a scheduling problem can have several constraints [15]. These constraints include but are not limited to the following:

- *Precedence constraint* Tasks may have precedence relations between them which can be represented by an acyclic directed graph $G = (V, E)$ where the vertices correspond to the tasks and an edge from vertex i to vertex j indicates that task i must be completed before task j can start processing.

- *Preemption* Processing of a task can be interrupted and resumed at a later time, even on another machine.

- *Deadline* A task cannot be scheduled after its deadline.

Furthermore, various objectives are considered while handling a scheduling problem, such as minimizing makespan, minimizing maximum lateness, profit maximization, cost minimization, total flow time [15].

The following is an example illustrating the three-field notation.

Example 2.1. The problem of scheduling tasks with arbitrary precedence constraints on m identical parallel machines to minimize the makespan can be represented as: $I|prec|C_{max}$, where I,$prec$, and C_{max} represents identical machines, the precedence relations between tasks, and the makespan minimization objective, respectively.

2.7 Related Research

In this thesis, we have studied the resource assignment problem in vehicular clouds that falls in the category of unrelated parallel machine scheduling problems. Despite a rich literature in machine scheduling [38] and grid computing [21], this problem has not been examined extensively yet. In this section, we discuss the previous works which are related to the task scheduling problems in vehicular clouds.

2.7.1 Parallel Machine Scheduling

Task scheduling problems are known to be *NP*-hard [36, 24, 52], so finding a polynomial time algorithm is as hard as any other *NP*-hard problem (e.g.- knapsack problem, traveling salesman problem etc). Therefore, approximate algorithms are studied which guarantee solutions close to the optimal solutions with lower polynomial complexity.

The job-shop scheduling problem is the most general of the scheduling problems where a set of tasks need to be scheduled on a set of machines with varying processing powers, while the objective is to minimize the total scheduling length. Graham *et al.* [28] first presented a greedy *list scheduling* algorithm for this problem that has a worst case performance of $(2-1/m)$, where m is the number of machines. This algorithm assigns the jobs as soon as there is machine availability to process them: whenever a machine becomes idle, then one of the remaining jobs is assigned to start processing on that machine. Further, Graham *et al.* [29] proposed another greedy algorithm known as *LPT* (longest processing time) schedule for the multiprocessor scheduling that has worst case performance bounded by $4/3$. The idea of the LPT algorithm is to sort all the jobs in decreasing order of processing times and then apply the list scheduling algorithm.

Scheduling problems are studied with different objective functions subject to various constraints. Sahni *et al.* [49] discussed dynamic programming based algorithms for scheduling the independent task to obtain optimal solutions for the problems such as - job sequencing with deadlines (JSD) , minimum finish time (MFT), optimal mean flow time (OMFT)

and weighted mean flow time (WMFT). He also presented ε-approximate algorithms for the JSD problem. Additionally, Horowitz and Sahni [32] presented a polynomial time-bounded approximation algorithm for fixed number of unrelated machines. Gonzalez, Ibarra and Sahni [26] proposed an algorithm for uniform parallel machines where jobs are ordered in nonincreasing processing times and assigned to the fastest machine. They have shown that for $m > 2$ the performance of the algorithm approaches $3/2$ as m increases. In [54] a 2-approximation algorithm is presented that uses a local search method for scheduling jobs on identical parallel machines with the goal of minimizing the total finishing time of all the jobs. In [27] a polynomial time algorithm is given for preemptively scheduling n jobs on m uniform machines within a deadline. Furthermore, Papadimitriou *et al.* [46] presented a polynomial time scheduling algorithm on m processors subject to a precedence constraint.

In [50] scheduling problems with machine availabilities are considered where the processors are available to execute jobs non-preemptively only during certain moments in time with the objectives of minimizing maximum and total completion time of all the jobs and minimizing maximum lateness.

More details on different parallel machine scheduling problems along with their solved results are shown in [30, 38, 39, 40].

As mentioned above, task scheduling is *NP*-hard and one of the techniques to solve any *NP*-hard problem is to reduce it into a well-known combinatorial optimization problem. We observed that the single task scheduling sub-problem for the resource assignment problem in vehicular clouds can be solved approximately by reducing it to a knapsack cover (KC) problem instance. Hence, the relationship between single task scheduling and KC is shown below which is followed by the previous works for solving the KC problem.

2.7.2 Relationship with Knapsack Cover Problem

The single task allocation on a set of unrelated machines (vehicles) sub-problem can be related with the knapsack cover (KC) problem which is a transformation of the classical

knapsack maximization problem into a minimization problem. Each machine (vehicle) can be considered as an item of the KC instance with a processing capacity (size) and a rental cost (cost) calculated based on an availability interval. Once the single task scheduling problem instance is constructed, the near-optimal solution can be found by applying the approximation algorithms proposed for solving the KC problem such that it satisfies all the constraints of the task scheduling problem in VC.

Following are some related works on the knapsack cover (KC) problem which can be used for solving the task allocation problem in vehicular clouds.

2.7.3 Approximation Algorithm for Knapsack Cover

The knapsack cover (KC) problem is the minimization version of the classical knapsack problem (KP). At first, an existing pseudo-polynomial time algorithm and a FPTAS for the knapsack problem is discussed. Further, we describe a $(1+\varepsilon)$ approximation scheme for the knapsack cover problem proposed by Islam [34].

PTAS and FPTAS for Knapsack problem Ibarra and Kim [33] proposed a dynamic programming (DP) based pseudo-polynomial time algorithm for the KP problem. They used the following recurrence to build the DP table where each item $i \in \{1,...,n\}$ represents a row and the profit values are the columns.

$$A(i,p) = \begin{cases} A(i-1,p) & \text{if } p_i > p \\ min\{A(i-1,p), w_i + A(i-1,p-p_i)\} & \text{Otherwise.} \end{cases}$$

Here, $A(i,p)$ denotes the total weight of the selected items. The subset of items with maximum profit and total weight within knapsack capacity C is considered as the optimal solution. Suppose P is the most profitable item among all the items, then this pseudo-polynomial time algorithm has a running time of $O(n^2P)$.

Further, using a scaling based technique the above mentioned pseudo-polynomial time algorithm can be modified into a FPTAS for the knapsack problem and can exihibit a performance ratio of $(1-\varepsilon)$ for all $\varepsilon > 0$ [33].

Algorithm 1: Pseudo-polynomial time algorithm for knapsack cover

Input : Set of items $\{1,...,n\}$ with each item having cost c_i and size s_i and an interger d.

Output: A minimum cost feasible solution.

1 Set $C = max\{c_i\}$.

2 Set $A(0,0) = 0$, $A(0,c) = \infty$ for $c = 1,...,nC$

3 **For** $i = 1,....,n$ and $c = 0,...,nC$ **do**

$$A(i,c) = \begin{cases} A(i-1,c) & \text{if } c_i > c \\ max\{A(i-1,c), s_i + A(i-1,c-c_i)\} & \text{Otherwise.} \end{cases}$$

4 Determine the minimum cost subset $S(i,c)$ such that $\sum_{i \in S(i,c)} s_i \geq d$ and return $S(i,c)$ as final solution.

FPTAS for Knapsack Cover Islam [34] proposed a scaling based fully polynomial time approximation scheme for minimum knapsack cover problem in his thesis. Initially they discussed a dynamic programming based pseudo polynomial time algorithm for the minimum knapsack problem. Given an instance of the KC problem with n items where each item $i \in \{1,...,n\}$ has a cost c_i and a size s_i, and a demand d. Let $S(i,c)$ denote a subset of items $\{1,...,i\}$ such that the total cost of these items is exactly c and total size is maximum. Furthermore, let $A(i,c)$ denotes the sum of the sizes of the items in subset $S(i,c)$. Therefore, the following recurrence can be used to compute $A(i,c)$,

$$A(i,c) = \begin{cases} A(i-1,c) & \text{if } c_i > c \\ \\ max\{A(i-1,c), s_i + A(i-1,c-c_i)\} & \text{Otherwise.} \end{cases}$$

The optimal solution can be found by selecting the minimum cost subset of items such that the total size of the items in this subset is at least d. Suppose $C = max\{c_i\}$ is the maximum cost item among all the n items. So the maximum cost of a solution can be nC. As there are n items, therefore total running time of the dynamic programming algorithm will be $O(n^2C)$ which is pseudo polynomial in n. The pseudocode for this is presented in Algorithm 1.

Now using a scaling technique a FPTAS can be obtained for the KC problem [34]. Consider a subproblem P_i that involves i number of items $\{1,2,...,i\}$. Suppose c_i^* is the maximum cost item in the subproblem. Therefore for any $\varepsilon > 0$, obtain the scaling factor

Algorithm 2: FPTAS for Minimum Knapsack

```
Sort all the items in nondecreasing order of their costs.
foreach i = 1 to n do
    foreach subproblem P_i do
        Let scaling factor, k_i = εc_i^* / i for a fixed ε > 0.
        For each item j, set scaled cost c'_j = ⌊c_j / k_i⌋.
        Apply the DP algorithm 1 with these scaled cost and find the minimum cost
        feasible solution.
    end
end
Find the minimum cost feasible solution over all the subproblems and output this as
the final solution.
```

k_i for this subproblem as: $k_i = \frac{\varepsilon c_i^*}{i}$ and the scaled cost of each item j in P_i as $c'_j = \left\lfloor \frac{c_j}{k_i} \right\rfloor$. Using these scaled cost apply algorithm 1 to find the minimum cost solution. Hence, the minimum cost feasible solution for the KC problem will be the minimum cost solution over all the subproblems. The procedure is given in Algorithm 2.

The following theorem is claimed and proved in the thesis [34] for this FPTAS.

Theorem 2.2. *[Islam et al., 2009] If A is a solution returned by the Algorithm 2 and O is the optimal solution then, $c(A) \leq c(O)(1+\varepsilon)$, where $c(V)$ is the total cost of items in set V and the running time of the algorithm is $O(\frac{n^4}{\varepsilon})$.*

Greedy Approximation Algorithm for Knapsack Cover Gens and Lenver [25] proposed a greedy heuristics for the knapsack cover problem. Further analysis of this greedy heuristic with the proof for the worst-case bound of 2 is described in [17].

Given a list $L = \{1,...,n\}$ of items. Let s_i and c_i denote the size and the cost, respectively, of an item i of the problem instance and D is the demand. Furthermore, relative cost of an item i is: c_i/s_i. Therefore, the steps for this greedy algorithm are as follows:

Step 1. Sort the the items in L such that $c_1/s_1 \leq c_2/s_2 \leq .... \leq c_n/s_n$.

Step 2. Find index k_1 such that $\sum_{i=1}^{k_1} s_i < D < \sum_{i=1}^{k_1+1} s_i$. Let $S_1 = \{1,2,..,k_1\}$ be the first set

of small items. Therefore, the candidate solution is $S_1 \cup \{k_1+1\}$.

Step 3. Find all the items before k_2 such that $\sum_{i=1}^{k_1} s_i + s_{k_2} < D$ and $\sum_{i=1}^{k_1} s_i + s_j \geq D$, for all $j \in [k_1+2, ..., k_2-1]$. Let $B_1 = \{k_1+1, ..., k_2-1\}$ as the first set of big items, then set $S_1 \cup \{j\}$, $j \in \{k_1+2, ..., k_2-1\}$ as candidate solutions.

Step 4. Find index $k_3 \geq k_2$ such that $\sum_{i=1}^{k_1} s_i + \sum_{i=k_2}^{k_3} s_i < D \leq \sum_{i=1}^{k_1} s_i + \sum_{i=k_2}^{k_3+1} s_i$. Let $S_2 = \{k_2, .., k_3\}$ be the second set of small items. Then $S_1 \cup S_2 \cup \{k_3+1\}$ becomes another candidate solution.

Step 5. Repeat from *step 2* to *step 4* until the end of the list L where in i-th iteration use k_{2i+1} instead of k_1 and k_{2i+2} instead of k_2.

Step 6. Return the smallest cost candidate solution.

In the above procedure, the sorting step has a computational complexity of $O(n \log n)$ and the later steps take $O(n)$ time. Suppose $GR(L)$ is the solution obtained from the above greedy procedure and $OPT(L)$ is the optimal solution. Following theorem and its proof is given for the worst-case bound of this greedy algorithm in [17].

Theorem 2.3. *[Csirik et al., 1991] For all lists L, $GR(L) \leq 2 \cdot OPT(L)$.*

Moreover, an improved bound of $3/2$ in $O(n^2)$ running time for the knapsack cover problem is also presented in [17].

2.7.4 Lower bound for Knapsack Cover

A greedy procedure is proposed by Gens and Lenver [25] that computes the lower bound for the knapsack cover problem. Suppose $S = \{1, 2, ..., n\}$ is the set of items with s_i and c_i be the size and cost, respectively, of each item i. Let lb denote the lower bound value. The description of this procedure is given in Algorithm 3 which has a running time of $O(n \log n)$.

In this thesis, we have extended the above discussed procedures for the task scheduling problem in vehicular cloud.

Algorithm 3: Lower bound for Knapsack Cover

1 Sort all the items according to the nondecreasing order of relative costs.
2 Set lower bound, $lb = \sum_{i=1}^{n} s_i$.
3 Fill the knapsack based on the sorted order of the items until k such that $\sum_{i=1}^{k} s_i \geq D$.
4 Set $C = \sum_{i=1}^{k} c_i$.
5 Set $lb \leftarrow min(lb, C)$.
6 Set $S \leftarrow S \backslash k$. If $\sum_{i \in S} s_i > D$, then repeat *step 3* to *step 5*.

Chapter 3

Resource Assignment in Vehicular Clouds

This chapter discusses the details of the problem in consideration along with the proposed solutions. At first, the system model is presented in section 3.1. Then the problem formulation and the proposed algorithms are discussed in sections 3.2 and 3.3, respectively. Finally, an algorithm with the guranteed accuracy for this problem is proposed in section 3.4.

3.1 System Model

We consider a vehicular cloud (i.e.- can be centralized or autonomous as described in 2.4.2) consisting of a set of m vehicles available during a determined time interval. An arbitrary vehicle is denoted by symbol i. Each vehicle i has a processing speed (processing capability) denoted by $\alpha(i)$. It represents the number of operations the processor can handle per unit of time. Since a vehicle will not stay in the area that defines the VC forever, it has an availability interval where the arrival and the departure time of vehicle i is defined by t_i^- and t_i^+, respectively. Additionally, based on i's resource capabilities a constant rental cost per unit of time, $C(i)$, is associated with vehicle i. Hence, the total resource utilization cost for vehicle i is calculated based on how long it's resources are going to be used if selected.

Each processing task is denoted by k and has a processing requirement ρ_k, measured by the number of machine instructions, which needs to be served by the resources in the vehicular cloud. We consider an offline environment where the problem instance is completely available that means the algorithm has access to the whole input instance before starting

the scheduling process. In this problem, the availability of the vehicles and the processing requirements of the tasks are known ahead of time. We assume that all the tasks have a common deadline T, representing the maximum allowable duration each task can take to finish. We note that this requirement can be made more general to include task specific deadlines.

Preemption is allowed to complete the processing requirement of each task. Situations for preemption in VC are described in Section 3.1.1.

3.1.1 VC-preemption

Processing of any task can be interrupted at any time and resumed immediately on another processor. We call this event *task migration*. In this study, we focus on the effect of availability of processors on the computational capabilities of the cloud and ignore the task transfer cost. A task is allowed to migrate to a different processor if the current processor becomes unavailable or if a more cost efficient processor is found.

3.1.2 Scheduling Problem in VC

We adapt the standard three field problem notation (see Section 2.6) to our context: $T/M/C$, where T is an integer representing the number of tasks to be scheduled, M represents the machine environment which is I for identical machines or U for unrelated machines, and C is the constraint which is VC for VC-preemption with task deadlines. Therefore, the single task scheduling and multitask scheduling problem in vehicular clouds can be represented as $1/U/VC$ and $n/U/VC$, respectively. The problem objective is to minimize the cost of the schedule.

For the task scheduling problem in vehicular cloud (VC), a schedule is called *feasible* if it satisfies the following constraints:

- All tasks must be processed completely. (In case of experimental analysis, this constraint is relaxed for the experimental scenario presented in Section 4.4.)

- All tasks must be processed before the deadline, T.
- (Migration constraint) A task is not allowed to be paused and resumed at a later time, it can only be transferred and resumed immediately on a different machine without any loss.
- Each machine can process only one task at a time.
- (Availability constraint) Each machine can only process during its availability time.

The formal description of a *schedule* and the *cost of a schedule* for the scheduling problem in VC is given below:

Schedule A task schedule consists of an assignment of tasks to processors at certain moments in time. If a task k is assigned to l_k different machines, then the schedule for task k specifies a list of l_k consecutive time intervals denoted $L_k = (\delta_1, \delta_2, \ldots, \delta_{l_k})$ and a corresponding sequence of machines $M_k = (i_1, i_2, \ldots i_k)$ so that task k is scheduled on machine i_s during the time interval δ_s. A time interval δ_s consists of a pair of time values, $\delta_s = (\delta_s^-, \delta_s^+)$ with $\delta_s^- < \delta_s^+$. The time intervals are consecutive, $\delta_s^+ = \delta_{s+1}^-$. If we abuse the notation to denote by δ_s the duration of time interval δ_s, then a feasible schedule must satisfy the task's demands, $\sum_{s=1}^{l_k} \delta_s \alpha(i_s) = \rho_k$, and if two tasks are scheduled on the same machine, then the time intervals during which the two tasks are scheduled are disjoint.

Cost of schedule Any schedule has a cost associated equal to the total cost of renting the resources, $\sum_{k=1}^{n} \sum_{i \in L_k} C(s)\delta_s$. The objective of the scheduling is to minimize this cost.

3.2 Resource Assignment in Vehicular Clouds

In this thesis, we have focused on a resource assignment problem in vehicular clouds that falls in the class of machine scheduling problems where the smart vehicles are considered as the machines with computing resources. In this section, at first we formulate the problem and then we describe our algorithms proposed for solving the problem.

3.2.1 Problem Formulation

We divide the required allocation interval (i.e.- the life time of a VC) into consecutive time intervals determined based on the availability change (e.g. arrival or departure) information of the vehicles. Each time interval $[t_1, t_2]$ is such that t_1 and t_2 represent the starting or the ending point of the availability interval for some vehicle and no other availability interval starts or ends between t_1 and t_2. In fact, the set of available vehicles does not change between t_1 and t_2. We call these intervals **partitions**. The available processors in each partition are called the **objects** and are denoted by o_{ij}.

Definition 3.1. An arbitrary partition P_i is a set of objects such that each object $o_{ij} \in P_i$ has a length of $\delta_i = Pt_i^+ - Pt_i^-$, where δ_i represents the partition length, Pt_i^- is the start time, and Pt_i^+ is the end time of partition P_i.

We denote the set of partitions and the set of objects as $\mathcal{P}$ and O, respectively. The end time of the last partition corresponds to the deadline T. Each object $o_j \in P_x$ has a resource capacity of s_{xj} and a total access cost c_{xj} which are calculated as: $s_{xj} = \alpha(j) * \delta_x$ and $c_{xj} = C(j) * \delta_x$, respectively. We assume that all the cost values are integers. Moreover, we introduce the notion of scheduling intervals, collectively represented by set I, where each interval I_i is defined as follows:

Definition 3.2. A scheduling interval I_i, denoted by $[Pt_s^-, Pt_t^+]$, is a collection of objects from partitions $\{P_s, P_{s+1}, ..., P_t\}$ where Pt_s^- and Pt_t^+ are the start time and the end time, respectively, of this interval. The scheduling interval length $|I_i|$ is measured as the number of partitions from P_s to P_t.

Example 3.3. Given two partitions, P_1 and P_2, so $I = \{I_1, I_2, I_3\}$ such that $I_1 = \{P_1\}$, $I_2 = \{P_1, P_2\}$ and $I_3 = \{P_2\}$ and length of each scheduling interval is $1, 2$ and 1, respectively.

Additionally, we represent the set of processing requirements as $\Gamma = \{\rho_1, \rho_2, ..., \rho_n\}$. The total demand of all the tasks is denoted by D, and calculated as $D = \sum_{k=1}^{n} \rho_k$.

Table 3.1 summarizes the notations used throughout the thesis.

Table 3.1: Notation Table

Symbol	Meaning
i	An arbitrarily vehicle.
$\alpha(i)$	Processing speed per unit of time of vehicle i.
$C(i)$	Cost per unit of time of vehicle i.
$[t_i^-, t_i^+]$	Availability interval of vehicle i.
k	An arbitrarily task.
ρ_k	The processing requirements of task k.
D	(Demand) Total processing requirements of all the tasks.
T	Common deadline for all the tasks.
Γ	The set of processing requirements.
P_x	An arbitrary partition x.
δ_x	The length of a partition P_x.
$\mathcal{P}$	The set of partitions.
o_{xj}	An object o_j of partition P_x.
s_{xj}	The size of an object o_{xj}.
c_{xj}	The cost of an object o_{xj}.
O	The set of objects.
I_y	An arbitrary interval y.
I	The set of intervals.

Figure 3.1 shows an example of the resource allocation problem formulation in a VC scenario.

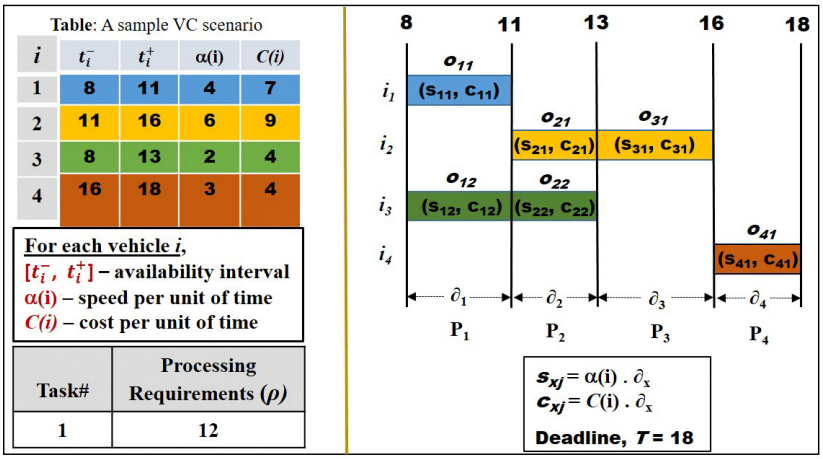


Figure 3.1: Resource allocation problem formulation in VC - an example.

NP-hardness result for $1/U/VC$: As mentioned in Section 2.7.2, the single task scheduling problem, represented as $1/U/VC$, is related to the knapsack cover (KC) problem. To prove that $1/U/VC$ optimization problem is *NP*-hard, we need to show that its corresponding decision problem is *NP*-complete. Decision version of the $1/U/VC$ problem is given below.

Problem 3.4. ***1/U/VC***: Given a set of m machines $V = \{1,2,\ldots,m\}$ with a capacity (size) $S_i \in \mathbb{Z}^+$ and a rental cost $C_i \in \mathbb{Z}^+$ for each machine i, a task with task requirement $\rho \in \mathbb{Z}^+$, and a scheduling cost $\mathcal{K} \in \mathbb{Z}^+$. Does there exist a schedule V' such that $\sum_{i \in V'} S_i \geq \rho$ and $\sum_{i \in V'} C_i \leq \mathcal{K}$?

Now, to prove that the decision version of $1/U/VC$ is *NP*-complete (see Section 2.5), at first, we show that it belongs to the class *NP* and then we show a polynomial time reduction from the KC problem which is a known *NP*-complete problem. The decision version of the KC problem is given below.

Problem 3.5. **Knapsack Cover (KC)**: Given is a set of n items $X = \{1,....,n\}$ with a cost $c_a \in \mathbb{Z}^+$ and a size $s_a \in \mathbb{Z}^+$ for each item a, a demand $D \in \mathbb{Z}^+$ and a value goal $\mathcal{K} \in \mathbb{Z}^+$. Is there a subset of items $F \subseteq X$ such that $\sum_{a \in F} s_a \geq D$ and $\sum_{a \in F} c_a \leq \mathcal{K}$?

Following is the formal proof for the *NP* completeness of $1/U/VC$.

Theorem 3.6. *Task migration constraint.* $1/U/VC$ *is NP-complete.*

Proof. $1/U/VC$ is in *NP*. Given a schedule consists of a set V' of machines, we can verify in polynomial time that the sum of their capacities (total size) is at least ρ, and the sum of their rental costs is at most $\mathcal{K}$.

Given an instance X of KC, we construct an instance V of $1/U/VC$ as follows. Each item $a \in X$ in KC corresponds to a machine i with an availability interval and speed calculated so that it equals the size of the item. The availability intervals are chosen so that they are consecutive in some arbitrary order of the items. Suppose, δ_{t_i} and $\alpha(i)$ are the length of the availability interval and the speed, respectively, of machine i such that its size S_i and

cost C_i, are set as $S_i = C_i = \delta_{t_i} * \alpha(i) = s_a = c_a$. Let A denotes the set of these machines. We set $\rho = \mathcal{K} = D$. We double the number of machines by adding exactly one machine with the availability interval equal to that of a machine corresponding to a KC item but with $(size, cost) := (0,0)$ and let B represents this set of machines. So, $V = A \cup B$. The construction idea is depicted in Figure 3.2. Clearly, this construction process is polynomial in the input size.

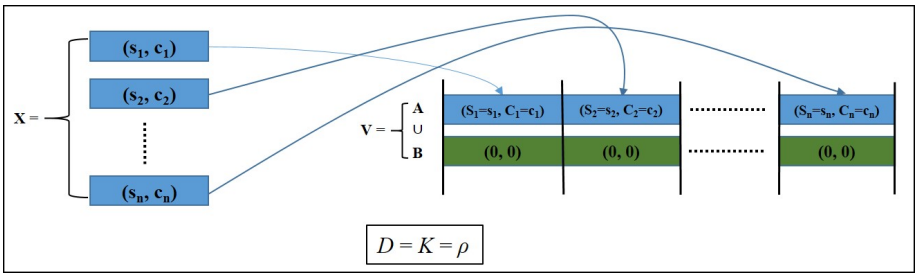


Figure 3.2: Construction of a single task scheduling problem instance V from a given knapsack cover (KC) instance X.

If X is a 'Yes' instance of KC problem, then we claim that V is also a 'Yes' instance of $1/U/VC$. Suppose $F \subseteq X$ is the set of items picked by the KC solution such that $\sum_{a \in F} s_a \geq D$ and $\sum_{a \in F} c_a \leq \mathcal{K}$. Let F corresponds to a feasible schedule for $1/U/VC$ such that $\sum_{a \in F} S_a + \sum_{b \in B} S_b = \sum_{a \in F} s_a \geq \rho$ and $\sum_{a \in F} C_a + \sum_{b \in B} C_b = \sum_{a \in F} c_a \leq \mathcal{K}$, that means the feasible schedule consists of machines corresponding to the items in the KC solution plus possible other machines with zero cost and speed that are chosen to satisfy the *VC-preemption* constraints. Therefore, V is a 'Yes' instance for the $1/U/VC$ problem.

Conversely, if V is a 'Yes' instance of $1/U/VC$ problem, then we claim that X is also a 'Yes' instance of KC. Suppose, there exists a subset $V' \subseteq V$ of machines in the schedule for $1/U/VC$ such that $\sum_{i \in V'} S_i \geq \rho$ and $\sum_{i \in V'} C_i \leq \mathcal{K}$. Let the KC solution contains the items in V', and it follows that $\sum_{i \in V'} s_i = \sum_{i \in V'} S_i \geq D$ and $\sum_{i \in V'} c_i = \sum_{i \in V'} C_i \leq \mathcal{K}$. We conclude that, X is a 'Yes' instance for the KC problem. □

Now, a scheduling interval corresponds to the actual period of time that a task is scheduled on the vehicular cloud processors. If the scheduling interval for a task is known, then

the actual minimum cost schedule for that task can be obtained by solving an instance of the multiple choice knapsack cover problem.

Problem 3.7. **Multiple Choice Knapsack Cover Problem (MCKCP)**: Given n classes $N_1, N_2, ..., N_n$ of items, where each item $j \in N_i$ has a non-negative size, s_{ij} and a non-negative cost c_{ij}, and given a demand $D \in \mathbb{Z}^+$, the objective is to choose exactly one item from each class such that the total size of the chosen objects is at least D and the total cost of the objects is minimized.

Hence, by incorporating the MCKCP constraint (i.e. pick exactly one item from each class) the $1/U/VC$ problem can be defined as below:

"*Given a set of intervals I and a task k with it's processing requirements $\rho_k \in \mathbb{Z}^+$, the objective is to find a minimum cost interval $I_i \in I$ subject to the constraint that exactly one object is chosen from each partition $P_i \in I_i$ and the total size of the chosen objects is at least ρ_k.*"

Since we do not know the optimal scheduling interval for task k, we can enumerate all possible scheduling intervals and we can solve the MCKCP corresponding to the scheduling interval, retaining the case with the smallest cost.

3.3 The Algorithms

3.3.1 Greedy Algorithm for $n/U/VC$

Scheduling for $n/U/VC$ can be approached in a greedy fashion. This is the "main procedure" that takes each task, one at a time, and schedules it with the minimum cost by calling a second procedure, the "single task scheduling procedure" which solves the $1/U/VC$ problem. This second procedure can be implemented with two algorithms, a PTAS and a greedy approximation. Depending on the algorithm used for scheduling one task, we call the algorithm for problem $n/U/VC$ as *GrPTAS* or *GrGr*, respectively.

The basic idea of this greedy algorithm is:

(i) For each task k solve the MCKCP instance using the PTAS (described in Section 3.3.2) or the Greedy algorithm (described in Section 3.3.3) to find the cheapest available interval that fulfills the task requirements of k.

(ii) Repeat step (i) until task requirements of all the tasks are completed. If an available interval is not found for any task to complete then there is no feasible solution for the problem.

3.3.2 PTAS for $1/U/VC$ problem

We assume that the scheduling interval is given and thus we need to solve an instance of MCKCP. A scaling based fully polynomial time approximation scheme (FPTAS) for minimum knapsack cover (KC) problem is proposed in a thesis by Islam [34]. Their proposal uses a dynamic programming (DP) approach for solving the KC problem. We extend this scheme to the MCKCP problem. We show that with our recurrence, the algorithm exihibits the performance ratio of $(1+\varepsilon)$.

At first we show that, the MCKCP problem can be solved in pseudo polynomial time. The algorithm uses dynamic programming approach that builds a dynamic programming (DP) table. We assume that the cost of objects in the MCKCP are all integer.

We denote the DP sub-problem by $S(i,c)$, which represents the maximum total size of objects from partitions $\{1,....,i\}$ given a budget c used to pay the costs of the objects. The DP sub-problem can be obtained recursively,

$$S(i,c) = \max_{o_{ij} \in P_i \wedge c_{ij} \leq c} \{S(i-1, c-c_{ij}) + s_{ij}\} \tag{3.1}$$

In this relation, we extend the total size of the objects selected by one more partition.

Suppose there are total z partitions and c_{max} is the highest cost of an object among all partitions, then the maximum cost for any solution can be $\mathcal{C}^* = zc_{max}$. If there are a total of n objects, then the total running time taken for building the DP table is $O(n\mathcal{C}^*)$. Once the DP table $S(,)$ is built, the optimal solution can be found by looking at the entry that

Algorithm 4: Dynamic Programming Algorithm for $1/U/VC$

Input : Integers C^*, ρ_k, objects set O with size (s_{ij}) and cost (c_{ij}) of each object.
Output: A minimum cost feasible solution.

1 Set $S(0,0) = 0$, $S(0,c) = \infty$ for $c = 1,...,C^*$

2 **For** $i = 1,....,z$; $c = 0,...,C^*$ and $j = 1,....,n$ **do**

$$S(i,c) = \begin{cases} 0 & \text{if } c_{ij} > c \\ \max S(i-1, c-c_{ij}) + s_{ij} & \text{Otherwise} \end{cases}$$

3 Determine the minimum cost feasible solution and return as final solution.

satisfies the demand with minimum cost c. This is a pseudo polynomial time algorithm for the $1/U/VC$ problem. A description of the steps of this dynamic programming algorithm is given in Algorithm 4.

Algorithm 4 returns the subset $O' \subseteq O$ with minimum cost $c \in \{0,...,C^*\}$ such that $\sum_{o_{ij} \in O'} s_{ij} \geq \rho_k$.

Now the above pseudo polynomial time algorithm can be converted into a fully polynomial time approximation scheme (FPTAS) using a cost scaling technique. For each partition P_l where $l \in [1..z]$, a sub-problem denoted by SP_l is solved, using the dynamic programming approach with object costs scaled by a scaling factor.

Definition 3.8. A sub-problem SP_l is a collection of all the objects belong to the partitions $\{P_1, P_2, ..., P_l\}$.

Suppose, c^* is the maximum cost object found in the optimal solution OPT. Since this value is unknown, for each object belongs to an interval I_l, we assume $c^* = c_{ij}$, and compute the scaling factor, denoted by k_l, as $k_l = \frac{\varepsilon c_{ij}}{z}$, for any $\varepsilon > 0$, where z is the interval length. Using this scaling factor k_l, cost of each object $o_{ij} \in I_l$ is scaled as follows:

$$c'_{ij} = \left\lfloor \frac{c_{ij}}{k_l} \right\rfloor$$

where c'_{ij} is the scaled cost of object o_{ij}. Using these scaled costs we apply the dynamic programming Algorithm 4 to find the minimum cost feasible solution in the space of scaled

Algorithm 5: FPTAS for $1/U/VC$

Input : A processing requirement ρ_k, intervals set I, objects set O with size (s_{ij}) and cost (c_{ij}) of each object and a number $\varepsilon > 0$.
Output: A minimum cost feasible schedule for task k.

```
foreach interval a ∈ I do
    Let O_a be the set of objects belong to a.
    foreach object cost c* ∈ O_a do
        Set scaling factor, k_l = εc*/l.
        Let O'_a ← {o_j ∈ O_a : c_j ≤ c*}.
        For each object o_j ∈ O'_a, set scaled cost c'_j = ⌊c_j / k_l⌋.
        Apply the DP algorithm 4 with these scaled cost on the instance (c*, ρ_k, O'_a)
        and let Z'(c*, a) be the minimum cost solution over the scaled costs.
        Let Z(c*, a) be the true cost corresponding to Z'(c*, a).
    end
end
Return min_{∀a∈I} { min_{∀c*∈a} {Z(c*, a)} }.
```

costs. This solution represents the task schedule for a fixed scheduling interval. This procedure is explained in Algorithm 5.

We claim the following theorem followed by the proof for the performance ratio of this algorithm.

Theorem 3.9. *For a fixed $\varepsilon > 0$ and m number of partitions, there is a FPTAS for the $1/U/VC$ problem that outputs a $(1+\varepsilon)$- approximation solution with running time $O(n^2 \frac{m^3}{\varepsilon})$.*

Proof. Let OPT be the optimal solution of the problem. Now consider a sub-problem SP_l consisting of the partitions $[P_1, P_2, ..., P_l]$. Suppose, S_l is the solution of the sub-problem SP_l returned by the dynamic programming algorithm after scaling the costs of the objects by the scaling factor k_l. For simplicity, we will use $k = k_l$ for further discussion. Let $C(S_l)$ denotes the sum of the original costs and $C'(S_l)$ denotes the sum of the scaled costs of the objects in set S_l.

As the dynamic programming algorithm returns the minimum cost solution, therefore

the following holds with the total scaled cost of the objects in the OPT denoted by $C'(OPT)$,

$$C'(S_l) \leq C'(OPT) \tag{3.2}$$

After scaling the costs by k, there is an error between the scaled cost of an object and the true cost of the object.

$$0 \leq c_{ij} - c'_{ij} \leq k \tag{3.3}$$

From equation 3.3 we can write,

$$C'(OPT) \leq C(OPT) \tag{3.4}$$

As there are l partitions in the sub-problem SP_l, so there will be a total of l objects in the solution S_l. Thus, the total error for this sub-problem is,

$$C(S_l) - C'(S_l) \leq kl \tag{3.5}$$

Now, with m partitions in the original problem, using equation 3.3, the total error in the optimal solution is,

$$C(OPT) - C'(OPT) \leq km \tag{3.6}$$

Assume that S is the final solution returned by the algorithm. So it is the minimum cost feasible solution over all the sub-problems, i.e. $C(S) \leq C(S_l)$. Therefore for a total of m partitions we can write,

$$C(S) - C'(S) \leq km \tag{3.7}$$

Adding equations 3.6 and 3.7,

$$C(OPT) - C'(OPT) + C(S) - C'(S) \leq 2km$$

$$
\begin{aligned}
\Rightarrow \mathcal{C}(S) &\leq 2km + \mathcal{C}'(OPT) + \mathcal{C}'(S) - \mathcal{C}(OPT) \\
&\leq 2km + \mathcal{C}'(OPT) + \mathcal{C}'(OPT) - \mathcal{C}(OPT) \quad \textit{using}(3.2) \\
&= 2km + 2\mathcal{C}'(OPT) - \mathcal{C}(OPT) \\
&\leq 2km + 2\mathcal{C}(OPT) - \mathcal{C}(OPT) \quad \textit{using}(3.4) \\
&\leq 2km + \mathcal{C}(OPT)
\end{aligned}
$$

Now, we choose k so that $2km \geq \varepsilon\mathcal{C}(OPT)$. If C^* is a lower bound on $\mathcal{C}(OPT)$, then $k = \frac{\varepsilon C^*}{2m}$. For example, we choose C^* to be the maximum cost of an object in OPT. Since this is unknown, we guess C^*.

$$\mathcal{C}(S) \leq 2\frac{\varepsilon}{2}\mathcal{C}(OPT) + \mathcal{C}(OPT)$$

$$\Rightarrow \mathcal{C}(S) \leq (1+\varepsilon)\mathcal{C}(OPT)$$

$$\Rightarrow \frac{\mathcal{C}(S)}{\mathcal{C}(OPT)} \leq (1+\varepsilon)$$

□

For the running time, with a total of m partitions and the highest cost C^*, maximum cost that occurs for the m^{th} sub-problem is mC^*. Now with the scaling factor $k = \frac{\varepsilon C^*}{2m}$ there will be $\frac{mC^*}{k} = \frac{2m^2}{\varepsilon}$ columns and m rows in the DP table for m^{th} sub-problem and total n calculations are required for n objects. So, total running time for this sub-problem is $O(n\frac{m^3}{\varepsilon})$. Moreover, the guess for the C^* is done with every possible object. Therefore the overall running time for a total of m sub-problems is $O(n^2\frac{m^3}{\varepsilon})$.

3.3.3 Greedy Algorithm for $1/U/VC$ problem

We propose a natural greedy algorithm to solve the $1/U/VC$ problem. The main idea is as follows: we enumerate all scheduling intervals. Given a fixed scheduling interval, we select the object from each partition with the smallest cost per unit of size value. We select the best feasible solution obtained over the set of all scheduling intervals.

Formal description of this procedure is given in Algorithm 6.

Algorithm 6: Greedy algorithm for $1/U/VC$

Input : A processing requirement ρ_k, intervals set I and objects set O with size (s_{ij}) and cost (c_{ij}) of each object.

Output: A minimum cost feasible schedule for task k.

```
foreach interval a ∈ I do
    foreach partition P_i ∈ a do
        Find the object with minimum c_ij / s_ij.
        Save s_ij of minimum ratio object. Save c_ij of minimum ratio object.
    end
    Save interval a with chosen object sizes such that it satisfies the task requirement.
    Save the total cost of this interval.
end
Return the minimum cost feasible solution among all the intervals.
if no min. cost solution found then
    no solution
end
```

3.4 Lower bound for $n/U/VC$

Algorithm 3 solves the knapsack cover (KC) problem fractionally which is proposed in [25]. We propose the following simple lower bound on the cost of the optimal task schedule for scheduling n tasks by solving knapsack cover fractionally. Consider all machines $i \in \{1,\ldots,n\}$ indexed in non-decreasing order of their cost per speed ratio ($\frac{C(i)}{\alpha(i)} \leq \frac{C(i+)}{\alpha(i+)}$ for all $1 \leq i \leq n-1$). This ratio represents the cost for serving a unit of demand. We denote each ratio as $\lambda_1 = \frac{C(1)}{\alpha(1)}$, and so on. Given a problem instance I, the total demand of I is denoted as D. Let δ_{t_i} is the length of the availability interval of machine i which is computed as $\delta_{t_i} = t_i^+ - t_i^-$. Then the cost of renting a machine is, $C_i = \delta_{t_i} * C(i)$ and the

Algorithm 7: Lower bound for $n/U/VC$

1 Sort all i's in nondecreasing order of their cost per speed ratios.
2 Find λ_k such that k is the smallest index subject to $\sum_{i=1}^{k} S_i \geq D$.
3 Caculate the total cost, $C'(I) = \sum_{i=1}^{k-1} C_i + (\frac{D-\sum_{i=1}^{k} S_i}{S_k}) * C_k$.
4 Output $C'(I)$.

size is, $S_i = \delta_{t_i} * \alpha(i)$. We calculate the cost of serving the total demand D by the cheapest resources. Let $C'(I)$ represents the total cost of the schedule for instance I. Clearly, the optimal schedule cannot do better than this. This procedure is described in Algorithm 7.

Chapter 4

Experimental Analysis

In this chapter, the performance of the greedy algorithms, presented in Chapter 3, are evaluated in diverse set of scenarios.

4.1 Experimental Scenarios and System Setup

We consider a typical VC formed at a parking lot (or parking garage/driveway/traffic jam etc.). Everyday, many vehicles are parked in a parking lot for several hours with each having different (or same) arrival and departure times, as vehicles do not stay in the parking lot forever. These vehicles are parked idle with ample unexploited computing resources where the resource capabilities (e.g.- processing speed per unit of time) of each vehicle are different. Vehicles with these amount of untapped resources are the perfect candidates for nodes for a cloud system. These resources can be rented out from the vehicle owners with appropriate incentives (e.g.- access cost-per-unit of time) based on their capabilities and can be utilized to carry out computing tasks offloaded to it by the service provider.

We evaluated our scheduling algorithms in three different scenarios:

- fixed number of vehicles and varying number of tasks,
- varying the ratio between tasks with a large demand (long tasks) and tasks with a small demand (short tasks) while maintaining the ratio between total demand and the total available processing resource constant, and
- task profiles extracted from real grid workload traces.

Test instances for the first two scenarios are generated randomly for experimental analysis. For the last scenario we have used real time data traces. For each vehicle the information generated are: the availability span $[t_i^-, t_i^+]$, speed-per-unit ($\alpha(i)$) and access cost-per-unit ($C(i)$). A processing requirement (ρ_k) is generated for each task. For the randomly generated instances, we used a normal distribution for the duration of the availability intervals for the machines.

We used a bi-modal distribution for task requirements to create sets of short and long tasks. We call a task short (long) if its request can be served at the average machine speed in time that is a fifth of (three times) the average duration of a partition (see Section 3.2.1). The intuition behind these assumptions is to observe the behaviour of the algorithms when tasks having longer processing requirements. More details about the experimental setup for each of these simulation scenarios are explaind in their respective sections.

Since no benchmark problem instances are available for the $n/U/VC$ problem we have compared the performance of our proposed algorithms relative to each other. We consider a greedy algorithm for solving the $n/U/VC$ problem. We obtain the approximation ratios of our proposed algorithms with respect to the lower bounds obtained from the greedy algorithm for both large and small size test instances.

Let $C_A(I)$ is the cost of the solution returned by algorithm A on instance I and $C'(I)$ is the value obtained from the lower bound algorithm, then the approximation ratio (AR) of algorithm A on instance I is:

$$AR(I) = \frac{C_A(I)}{C'(I)} \geq 1. \tag{4.1}$$

We implemented our algorithms in the C++ programming language. All the computational experiments for the solved instances were carried out on an Intel(R) Core(TM) i7-4500U CPU@1.80 GHz x 4 with 8 GB RAM computer powered by the ubuntu 14.04 LTS operating system.

4.2 Scenario A: fixed number of vehicles and varying number of tasks

In this experiment, we observe the behaviour of the proposed approaches by varying the total demand while keeping the total resource capacity fixed. For varying the total demand we vary the number of tasks (n) with a fixed ratio of short and long tasks while the number of vehicles (m) participate to form a VC is kept fixed.

We categorized the test instances into two groups: small and large, based on the values of m (the number of vehicles) and n (the number of tasks). For both sizes of test instances, we evaluate the performance of the algorithms based on their calculated "approximation ratios".

4.2.1 Experimental settings

Randomly generated test instances are used for this experiment by varying the values for m and n. For small-size instances m and n values are taken as $\{5,8,10\}$ and $\{5,10,20,30,50,100,200\}$, respectively. For large-size instances $m \in \{20,30,50\}$ and $n \in \{10,20,30,50,100,200\}$ are considered. Therefore, 50 x 200 is the largest instance size that is used for the experiment.

For each m value, instances are generated based on the following factors:

- Availability intervals $[t_i^-, t_i^+]$. For each of the m vehicles the avilability interval is generated within a specific deadline, where t_i^- and t_i^+ are both integers from the normal distribution with an average interval length and a standard deviation.

- Speed-per-unit ($\alpha(i)$). For this scheduling problem, $\alpha(i)$ of each vehicle i is an integer number drawn from the uniform distribution $[1,10]$.

- Cost-per-unit ($C(i)$). The rental cost-per-unit value is an integer which is generated by uniform distribution $[1,10]$.

For each n value the following information is generated:

- Processing requirements (ρ). For all the n tasks processing requirements are generated with a fixed proportion of short and long tasks (i.e.- 90% short and 10% long). Definitions of short task and long task are given in the Section 4.1. All the processing requirements are integers and generated using a bi-modal distribution where the average processing requirements used for the short and long tasks are calculated based on their definition.

We generate 10 random machine instances for each value of m and 10 random sets of tasks for each value of n. We solved instances corresponding to all possible combinations of values for m and n and we report the average over all of the instances with the same value of n. So, it resulted a total of 100 test instances for a specific m value. For all the m and n values from the small and large groups (defined above), problem instances are generated following the similar procedure. The code for input data generation is implemented in Octave.

4.2.2 Results

Table 4.1 lists the performance ratio of the algorithms obtained for small-size test problems. Each ratio value entry in table 4.1 represents an average value based on 100 test instances generated for the same m and n. For all the instances ratio values are obtained for the GrGr and the GrPTAS with 3 different ε values with respect to the solution obtained from the lower bound (LB) algorithm for $n/U/VC$. Percentage of instances solved for each (m,n) pair are given in the "Solved Instances" column. Value 0 in this column means the algorithms were unable to find any feasible schedule for that (m,n) pair.

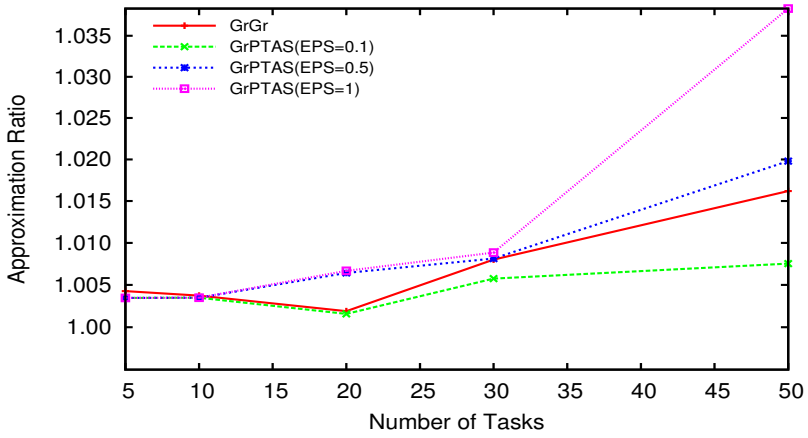
Figure 4.1 shows the "approximation ratio" as a function of number of tasks for small size test instances. Each line in this figure represents the algorithm used for solving the $1/U/VC$ instance. We observe that, GrPTAS (for $\varepsilon = 0.1$) performs better than GrGr for most of these small test instances.

Running times of the algorithms on small-size instances are given in table 4.2. Each

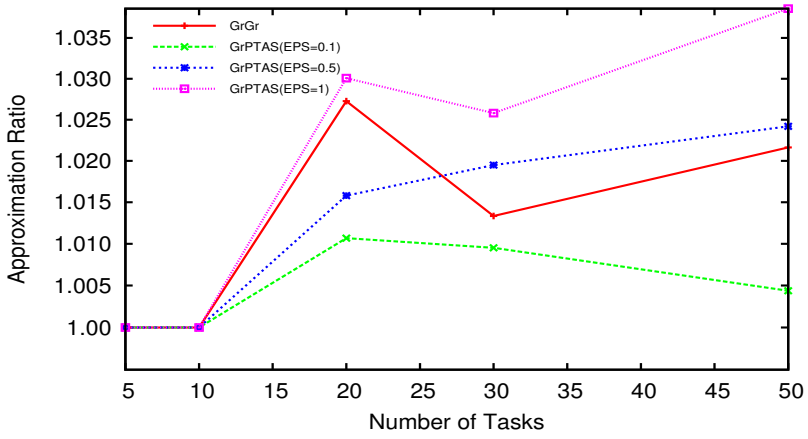
Table 4.1: Performance ratio for small test problems

m	n	Solved Instances (%)	Approximation Ratio			
			GrGr	GrPTAS		
				$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$
5	5	100	1.0042651763	1.0034562636	1.0034562636	1.0034562636
	10	100	1.0037292248	1.0034862914	1.0034862914	1.0034862914
	20	100	1.0018969333	1.0015681315	1.0064242807	1.0066708821
	30	70	1.0080097386	1.0057548386	1.0081280415	1.0088495752
	50	20	1.0161997705	1.0075369441	1.0197927653	1.0383268234
	100	0	NR	NR	NR	NR
	200	0	NR	NR	NR	NR
8	5	100	1	1	1	1
	10	100	1	1	1	1
	20	100	1.0272647209	1.0106707629	1.0157910906	1.0300768049
	30	100	1.0133409029	1.0095198702	1.0194901325	1.0258031901
	50	100	1.0216276729	1.0043784095	1.0241911958	1.0386142026
	100	0	NR	NR	NR	NR
	200	0	NR	NR	NR	NR
10	5	100	1	1	1	1
	10	100	1.0020834133	1	1	1
	20	100	1	1	1	1
	30	100	1.0036079245	1	1.0028944582	1.0203494529
	50	100	1.0006327535	1.0002571211	1.0036433571	1.0939219627
	100	100	1.0003333686	1.0010825887	1.0048393962	1.0453044202
	200	80	1.0023528692	1.0045015189	1.0128820598	1.0252636296

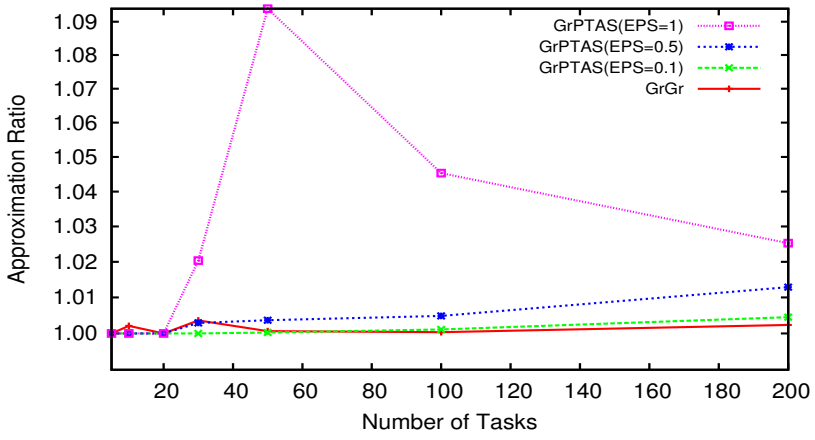
NR: No Result. No feasible schedules were found for these instances.



(a) Approximation ratios w.r.t. the number of tasks for $m = 5$



(b) Approximation ratios w.r.t. the number of tasks for $m = 8$



(c) Approximation ratios w.r.t. the number of tasks for $m = 10$

Figure 4.1: Performance comparison between *GrGr* and *GrPTAS* (for $\varepsilon = \{0.1, 0.5, 1\}$) in the scenario of fixed number of vehicles and varying number of tasks on small test instances for $m = \{5, 8, 10\}$.

Table 4.2: Time comparison for small test problems

m	n	Running Time (seconds)			
		GrGr	GrPTAS		
			$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$
5	5	.0001	0.015	0.005	0.003
	10	0.0002	0.03	0.005	0.004
	20	0.0003	0.048	0.008	0.006
	30	0.0005	0.09	0.012	0.008
	50	0.0008	0.20	0.018	0.015
	100	NR	NR	NR	NR
	200	NR	NR	NR	NR
8	5	0.05	0.08	0.013	0.009
	10	0.12	0.14	0.03	0.01
	20	0.25	0.34	0.07	0.03
	30	0.38	0.58	0.13	0.06
	50	0.64	1.008	0.21	0.11
	100	NR	NR	NR	NR
	200	NR	NR	NR	NR
10	5	0.0001	0.49	0.10	0.05
	10	0.12	0.60	0.12	0.06
	20	0.24	2.22	0.45	0.26
	30	0.37	3.52	0.70	0.36
	50	0.64	5.84	1.17	0.59
	100	1.29	12.64	2.54	1.29
	200	2.64	15.42	4.19	1.98

time entry in table 4.2 shows the averaged value over 100 instances measured in seconds. NR represents no result that means no feasible schedule were found for these instances.

Table 4.3 and 4.4 show the performance ratio and running times (in seconds), respectively, of the algorithms calculated for the large-size test problems. Each table entry represents an averaged value based on 100 instances with same (m,n) pair.

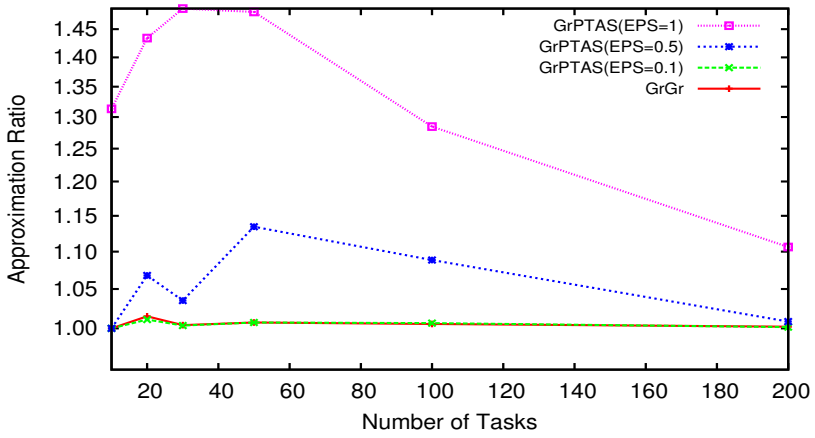
Figures 4.2 and 4.3 depict the comparison of approximation ratios and average running times (in seconds) for the large-size problem instances, respectively.

Table 4.3: Performance Ratio for large test problems

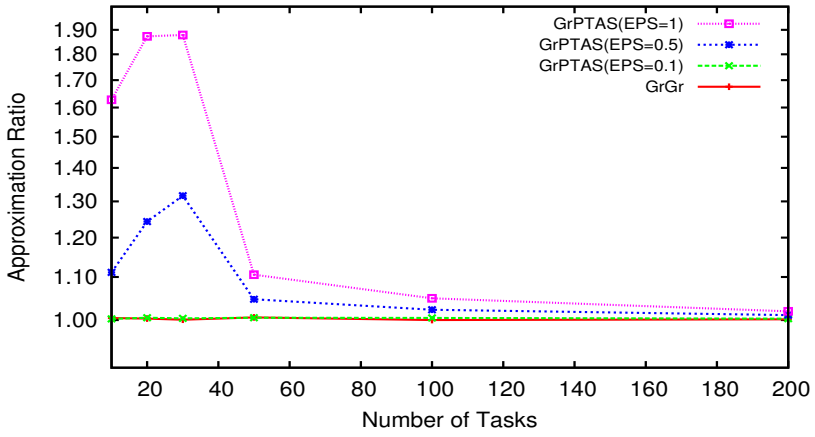
m	n	Approximation Ratio			
		GrGr	GrPTAS		
			$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$
20	10	1	1	1	1.3132884135
	20	1.0149864195	1.0110016632	1.0678738513	1.433949878
	30	1.0040286289	1.0034930499	1.0350509619	1.4874533071
	50	1.0072546277	1.007306224	1.1344033173	1.4812007261
	100	1.005412322	1.0064493545	1.08854351	1.2847682554
	200	1.0020657922	1.0014951151	1.008519394	1.1063117439
30	10	1.0044452002	1.0027206257	1.1113489769	1.6278264279
	20	1.0034145	1.0052614353	1.2433206347	1.8727646427
	30	1.0006978206	1.0036604889	1.3165311294	1.8786351984
	50	1.0059853691	1.005184444	1.0471565037	1.1056036436
	100	1.0000334124	1.0047810162	1.0229938326	1.0493419268
	200	1.0017240687	1.0030011488	1.0109509785	1.0195261929
50	10	1.0078810959	1	1.0088860679	1.8699196022
	20	1.0000286008	1.0000286008	1.2402299508	1.6370666972
	30	1	1	1.3806954008	1.5637541444
	50	1.0021928347	1.0057508991	1.3185778668	1.7470751582
	100	1.0028497485	1.0080463057	1.1868062582	1.4629466708
	200	1.0012647793	1.0028718395	1.1605930911	1.4237349384

Table 4.4: Time comparison for large test problems

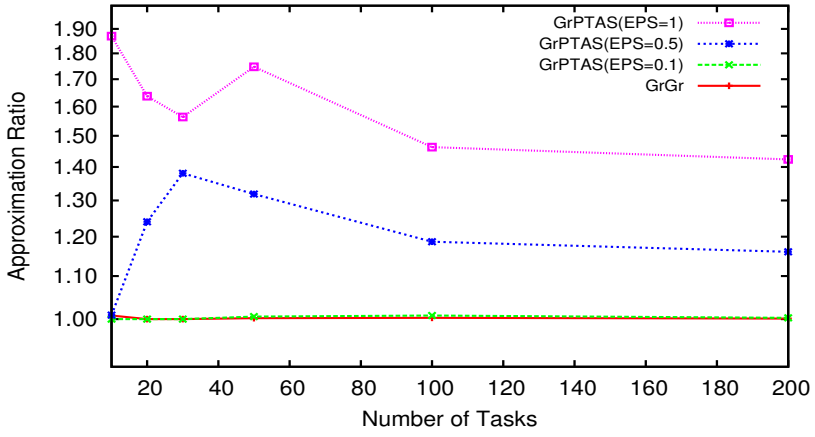
m	n	Running Time (seconds)			
		GrGr	GrPTAS		
			$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$
20	10	0.12	11.35	2.26	1.13
	20	0.25	23.81	4.73	2.35
	30	0.37	27.84	5.57	2.76
	50	0.62	68.15	13.58	6.75
	100	1.12	160.94	23.82	14.65
	200	2.57	259.39	52.19	26.37
30	10	0.12	194.64	38.85	19.38
	20	0.25	402.88	80.49	40.13
	30	0.37	663.87	132.45	66.13
	50	0.64	1212.20	510.47	380.03
	100	1.31	2606.83	1629.63	977.92
	200	2.65	5402.75	3788.76	2752.95
50	10	0.19	3301.98	658.25	328.23
	20	0.22	6325.78	1259.52	628.43
	30	0.37	8816.16	1742.97	869.88
	50	0.65	14495.70	2877.63	1436.29
	100	1.32	29136.60	5769.93	2878.12
	200	2.69	62017.96	12285.71	6122.81



(a) Approximation ratios w.r.t. the number of tasks for $m = 20$

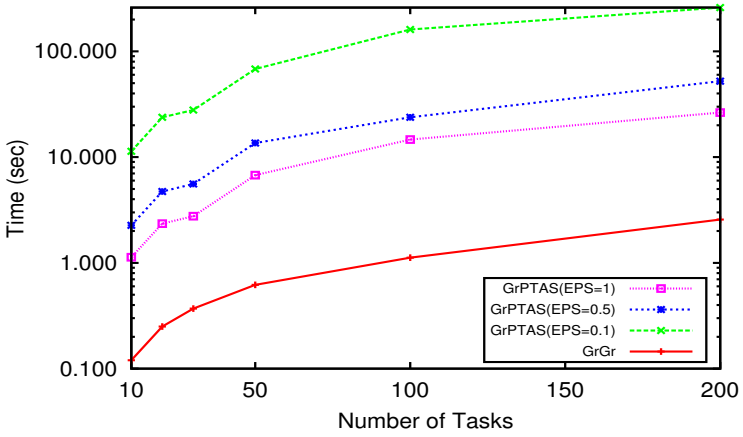


(b) Approximation ratios w.r.t. the number of tasks for $m = 30$

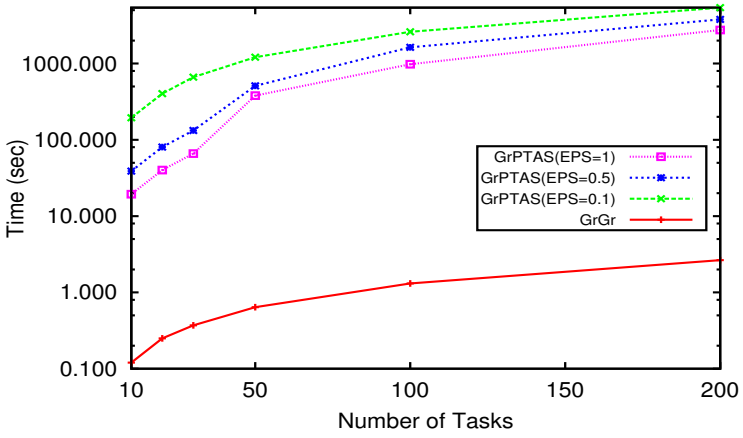


(c) Approximation ratios w.r.t. the number of tasks for $m = 50$

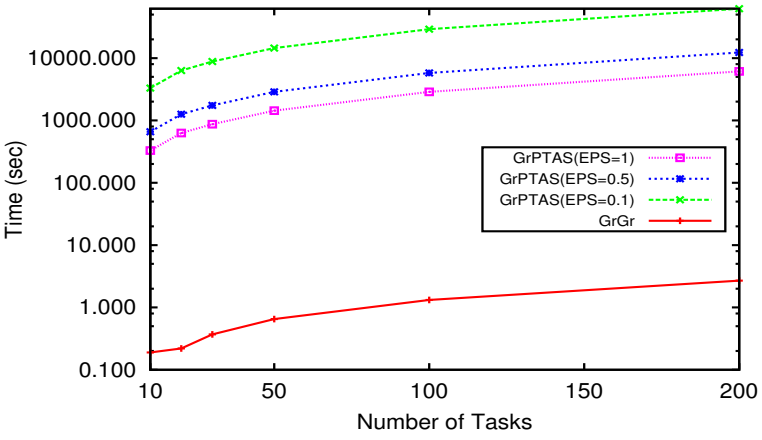
Figure 4.2: Performance comparison between *GrGr* and *GrPTAS* (for $\varepsilon = \{0.1, 0.5, 1\}$) in the scenario of fixed number of vehicles and varying number of tasks on large test instances for $m = \{20, 30, 50\}$.



(a) Average running time w.r.t. the number of tasks for $m = 20$



(b) Average running time w.r.t. the number of tasks for $m = 30$



(c) Average running time w.r.t. the number of tasks for $m = 50$

Figure 4.3: Average running time comparison between *GrGr* and *GrPTAS* (for $\varepsilon = \{0.1, 0.5, 1\}$) in the scenario of fixed number of vehicles and varying number of tasks on large test instances for $m = \{20, 30, 50\}$.

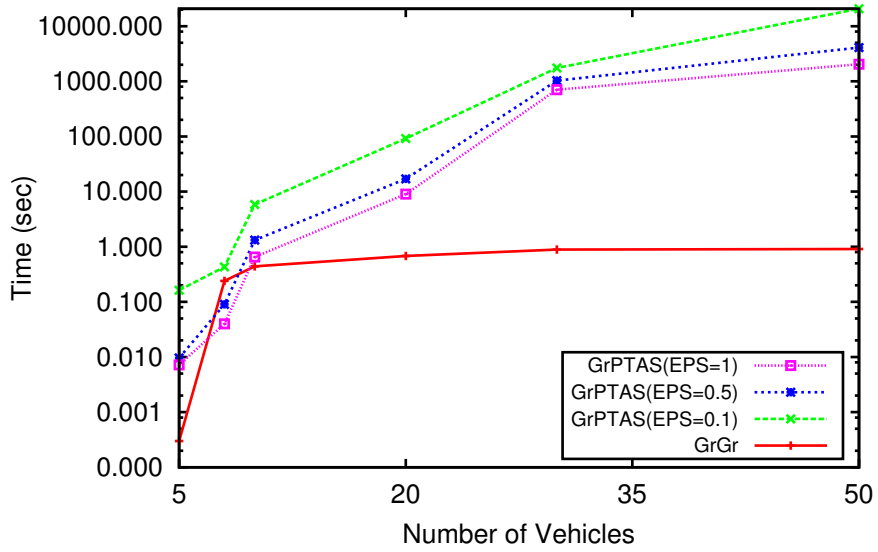
Table 4.5: Average running times with respect to the number of vehicles

m	Average Running Time (seconds)			
	GrGr	GrPTAS		
		$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$
5	0.0003	0.16	0.009	0.007
8	0.24	0.43	0.09	0.04
10	0.44	5.82	1.32	0.65
20	0.68	91.92	17.03	9.002
30	0.89	1747.19	1030.11	706.09
50	0.91	20682.36	4099.001	2043.96

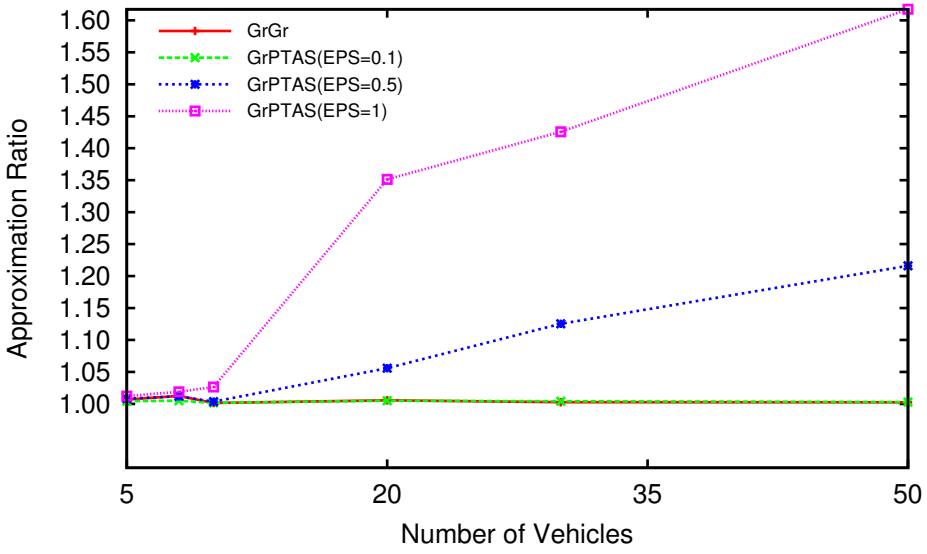
In table 4.5, the changes of the average running times of the algorithms for different number of tasks (i.e. number of tasks those are considered for the large and small test instances) for a fixed number of vehicles are given for the comparison purpose.

Figure 4.4 shows the performance of the algorithms when the number of vehicles increases. More specifically, Figure 4.4(a) depicts the average running times as a function of number of vehicles for the GrGr algorithm and for all three ε values of GrPTAS. Figure 4.4(b) shows the approximation ratios of the algorithms when the number of vehicles increases.

Observation It is clear that, both the *GrGr* and the *GrPTAS* (for $\varepsilon = 0.1$) shows similar performances in case of cost minimization. More specifically, *GrPTAS* performs noticeably well in case of small size test instances and sometimes better or similar performance for large test instances. On the other hand, *GrGr* outperforms *GrPTAS* in terms of running time for both size test instances.



(a) Average running time w.r.t. the number of vehicles.



(b) Approximation ratios w.r.t. the number of vehicles.

Figure 4.4: Performance comparison between *GrGr* and *GrPTAS* (for $\varepsilon = \{0.1, 0.5, 1\}$) as a function of number of vehicles.

4.3 Scenario B: Varying ratios with fixed constraint level

In this scenario we vary the ratio between the total processing demand of the tasks and the total processing capability of the machines. We call this ratio the *constraint level* (also known as *load factor*) and we denote it by γ.

4.3.1 Configuration of test problems

The problem instances for this experiment are generated based on following factors:

1. Constranit level (γ). We used five different constraint levels: 10%, 25%, 50%, 75% and 100%.

2. Number of vehicles (m). We used four different numbers: $8, 10, 15$ and 20. For each vehicle i, the availability interval, speed-per-unit and cost-per-unit are generated similarly as described in section 4.2.

3. Long/Short Ratio (β). We used task instances for five different ratios β between the total demand originating from long tasks and the total demand originating from short tasks for $\beta \in \{9, 1, 2, 1/2, 1/9\}$.

4. Number of tasks (n). The task instances were generated so that the constraint level remains constant. The number of tasks for this experiment is generated based on the β value and the total processing requirements are calculated based on the γ value. The calculations for generating n value and the processing requirements are shown below.

The following notations are used:

i) CAP_m : total processing capacity of m vehicles,

ii) PR_n : total processing requirements of n tasks,

iii) γ : constraint level of the problem,

iv) n_L and n_S : number of long and short tasks, respectively,

v) β : ratio of the number of long tasks to the number of short tasks,

vi) LPR_{avg} and SPR_{avg} : average processing requirement for long and short task, respectively,

vii) P_{avg} and S_{avg} : average partition duration and average speed-per-unit, respectively.

For any m, we know the values of CAP_m, P_{avg} and S_{avg}. Given a fixed value for γ, we can find the PR_n value as follows:

$$PR_n \ = \ \gamma \text{ x } CAP_m \tag{4.2}$$

Now, using the definition of short and long task we can compute the SPR_{avg} and LPR_{avg} values as below-

$$SPR_{avg} = \frac{P_{avg} \text{ x } S_{avg}}{number\ of\ short\ tasks\ per\ partition}, \tag{4.3}$$

where it is assumed that, the *number of short tasks per partition* may vary between 5 to 10. And,

$$LPR_{avg} = P_{avg} \text{ x } S_{avg} \text{ x } number\ of\ partitions\ per\ task, \tag{4.4}$$

where the *number of partitions per task* can be 3 or more. With PR_n, SPR_{avg} and LPR_{avg} calculated, we can write-

$$SPR_{avg} \ \cdot \ n_S + LPR_{avg} \ \cdot \ n_L = PR_n, \tag{4.5}$$

For a given β value, we know,

$$\begin{aligned} \beta &= \frac{n_L}{n_S} \\ \Rightarrow n_L &= \beta \ \cdot \ n_S \end{aligned} \tag{4.6}$$

Replacing n_L in equation 4.5 we get,

$$\begin{aligned} SPR_{avg} \cdot n_S + LPR_{avg} \cdot \beta \cdot n_S &= PR_n \\ \Rightarrow n_S(SPR_{avg} + \beta \cdot LPR_{avg}) &= PR_n \\ \Rightarrow n_S &= \frac{PR_n}{SPR_{avg} + \beta \cdot LPR_{avg}} \end{aligned} \tag{4.7}$$

Using 4.7 in 4.6 we get the value of n_L. So, total number of tasks generated are:

$$n = n_S + n_L$$

Finally, the processing requirement for each task is generated using a bi-modal distribution with the calculated average values for short and long tasks.

For each m value we considered all the five constraint levels while testing the performance of the proposed algorithms. For each (γ, β) pair, 10 random instances were generated. So, for a specific γ we generated 10 x 5 = 50 test instances. In total, 50 x 5 = 250 test problems were tested for each m value.

4.3.2 Computational results

Table 4.6 shows the setup used for generating test problems for this experiment. First three column entries show the vehicle information that includes - number of vehicles (m), average partition duration (P_{avg}) in seconds and average speed-per-unit (S_{avg}), respectively. Rest four columns represent the average processing requirements and the standard deviation value used for generating short and long tasks, respectively, using normal distribution. Here we assumed that, the *number of short tasks per partition* = 5 and the *number of partitions per task* = 3 while calculating the SPR_{avg} and LPR_{avg}, respectively.

Tables 4.7, 4.8, 4.9 and 4.10 list the performance ratio of the algorithms for each of the five γ values for $m = 8, 10, 15$ and 20, respectively. Each of the entry in these tables represent the averaged value generated for all β values under each γ. So, total 100 instances

Table 4.6: Experimental setup for test problems

m	P_{avg}	S_{avg}	Short Tasks		Long Tasks	
			SPR_{avg}	SPR_{SD}	LPR_{avg}	LPR_{SD}
8	102.86	4.50	92.57	46.29	1388.57	520.71
10	59.47	4.60	54.72	27.36	820.74	307.78
15	57.6	4.87	56.06	28.03	840.96	315.36
20	53.33	4.05	43.20	21.60	648.00	243.00

Table 4.7: Approximatio ratio for different constraint levels on $m = 8$

γ	Approximation Ratio			
	GrGr	GrPTAS		
		$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$
10%	1	1	1	1
25%	1.0385798697	1.0151462084	1.031272599	1.0410139468
50%	1.025316028	1.0145165543	1.0284017812	1.0389316325
75%	1.0566296818	1.0387399686	1.0607249695	1.0756284775
95%	1.0357084617	1.0193400494	1.0299195342	1.0454995954

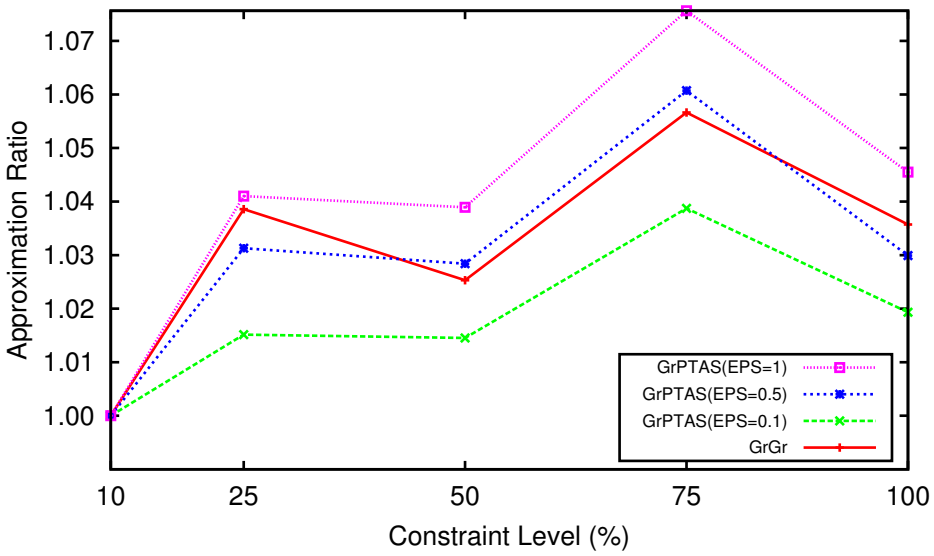
were tested for each γ.

Figures 4.5(a) and 4.5(b) depict the performance ratio and the average running time graphically for different load conditions for $m = 8$.

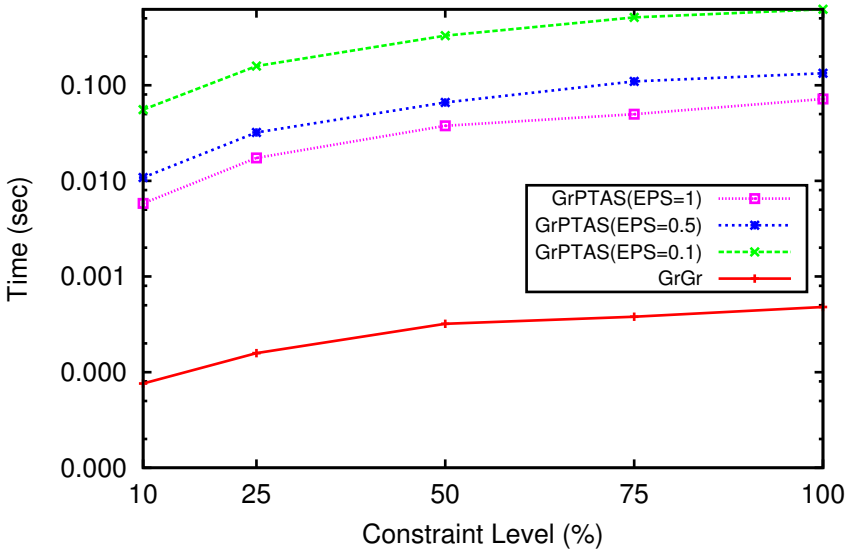
Figures 4.6(a) and 4.6(b) shows the performance ratio and the average running time for the algorithms in graphical format when $m = 10$.

Performance of the algorithms for $m = 15$ are shown in figure 4.7.

Figure 4.8(a) shows the performance ratio of the algorithms in graphical form and figure 4.8(b) shows the average running time taken by the algorithms for $m = 20$.



(a) Approximation ratios w.r.t different constraint levels for $m = 8$



(b) Time comparison w.r.t different constraint levels for $m = 8$

Figure 4.5: Performance comparison between *GrGr* and *GrPTAS* (for $\varepsilon = \{0.1, 0.5, 1\}$) in the scenario of different constraint levels with fixed number of vehicles for $m = 8$.

Table 4.8: Approximatio ratio for different constraint levels on $m = 10$

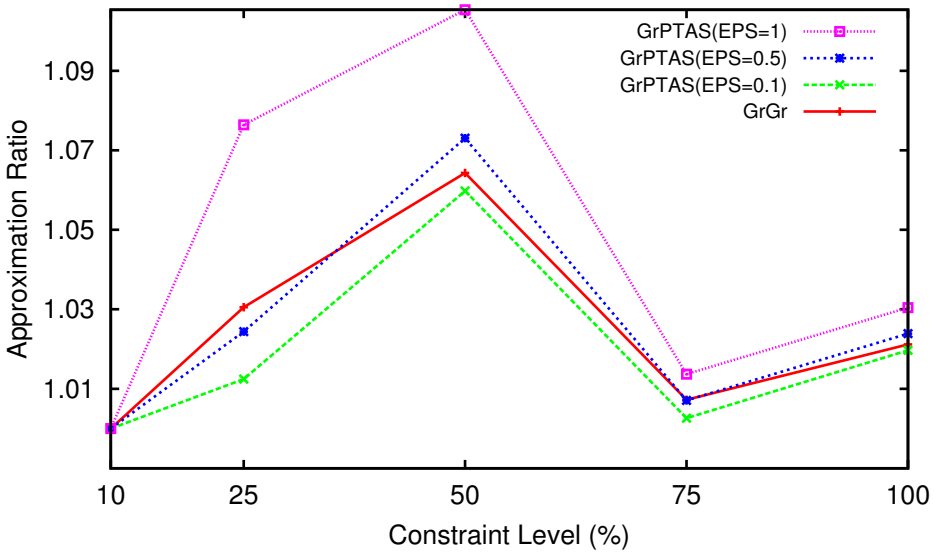
γ	Approximation Ratio			
	GrGr	GrPTAS		
		$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$
10%	1	1	1	1
25%	1.0305095661	1.0124226392	1.0243942757	1.0764115094
50%	1.0642984648	1.0597793637	1.0730401301	1.105347783
75%	1.0072355315	1.0026085114	1.007077421	1.0136714425
95%	1.0211724245	1.0196975725	1.0238819093	1.0304208273

Table 4.9: Approximatio ratio for different constraint levels on $m = 15$

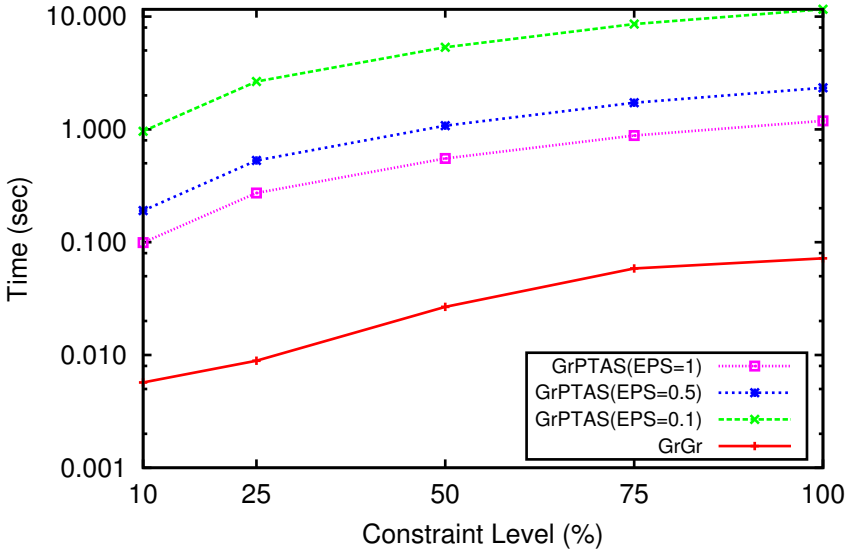
γ	Approximation Ratio			
	GrGr	GrPTAS		
		$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$
10%	1.0052729237	1.0055174534	1.3136787274	1.4692090996
25%	1.0014205901	1.0027045368	1.1834681644	1.3010851058
50%	1.0086692605	1.0082995969	1.0684859159	1.15107655
75%	1.0388831751	1.0197370174	1.0452380217	1.0726919522
95%	1.0143503292	1.0103138965	1.018375467	1.0315704628

Table 4.10: Approximatio ratio for different constraint levels on $m = 20$

γ	Approximation Ratio			
	GrGr	GrPTAS		
		$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$
10%	1.0215477658	1.0216786372	1.0421389202	1.2905405478
25%	1.0259012126	1.0213817908	1.0436631956	1.1090164646
50%	1.0120256336	1.0106101233	1.012455491	1.0335111287
75%	1.0073288192	1.0075112079	1.0113429221	1.0207583508
95%	1.0259461391	1.0259622386	1.0291551779	1.0451084967

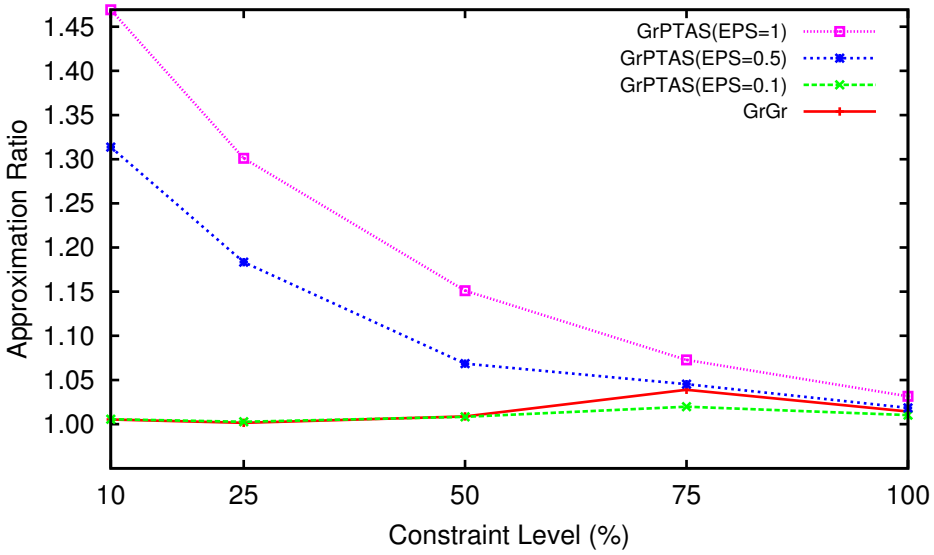


(a) Approximation ratios w.r.t different constraint levels for $m = 10$

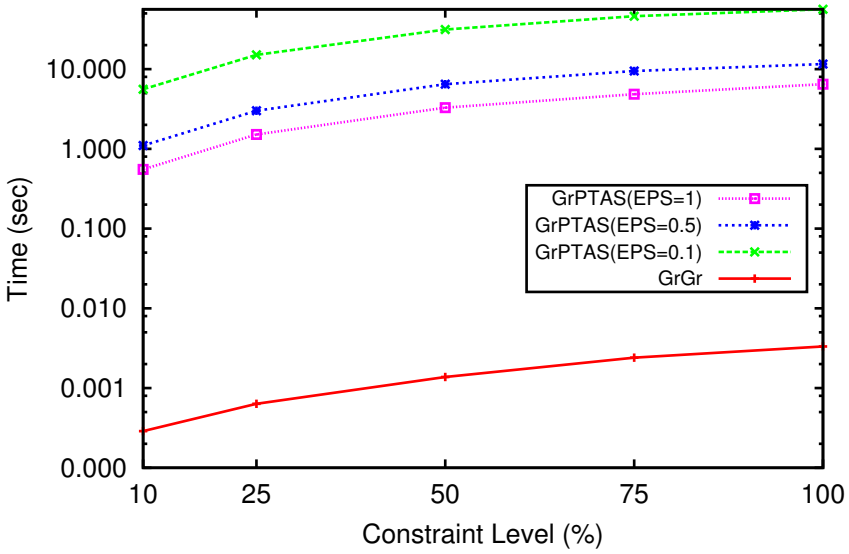


(b) Time comparison w.r.t different constraint levels for $m = 10$

Figure 4.6: Performance comparison between *GrGr* and *GrPTAS* (for $\varepsilon = \{0.1, 0.5, 1\}$) in the scenario of different constraint levels with fixed number of vehicles for $m = 10$.

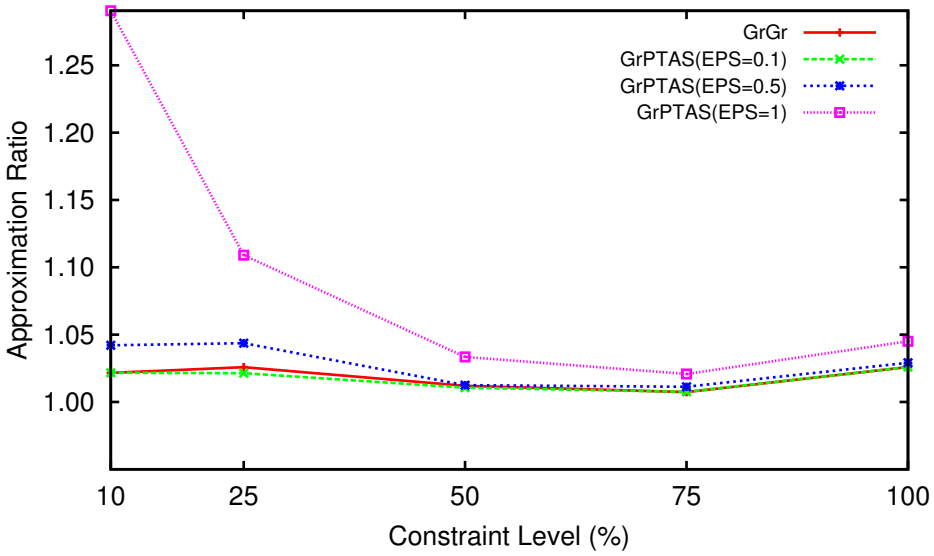


(a) Approximation ratios w.r.t different constraint levels for $m = 15$

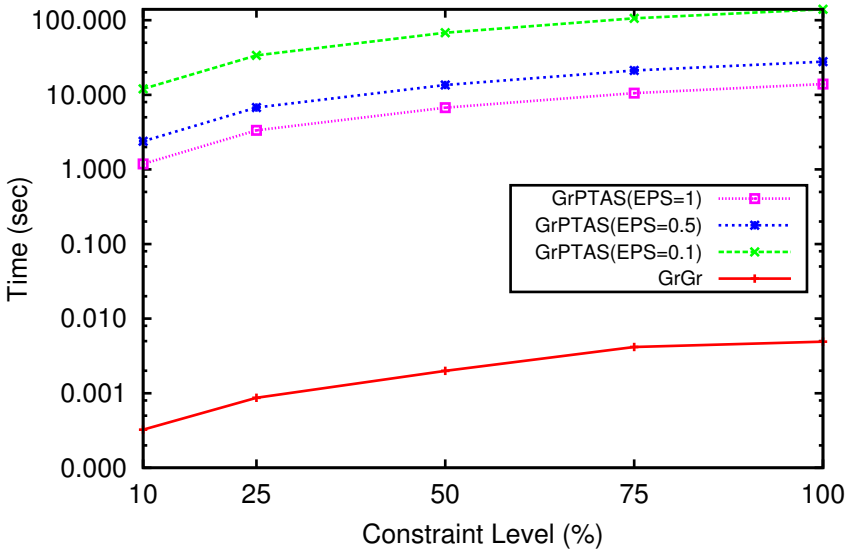


(b) Time comparison w.r.t different constraint levels for $m = 15$

Figure 4.7: Performance comparison between *GrGr* and *GrPTAS* (for $\varepsilon = \{0.1, 0.5, 1\}$) in the scenario of different constraint levels with fixed number of vehicles for $m = 15$.



(a) Approximation ratios w.r.t different constraint levels for $m = 20$



(b) Time comparison w.r.t different constraint levels for $m = 20$

Figure 4.8: Performance comparison between *GrGr* and *GrPTAS* (for $\varepsilon = \{0.1, 0.5, 1\}$) in the scenario of different constraint levels with fixed number of vehicles for $m = 20$.

Observation In terms of cost minimization, we observe that performance of the *GrPTAS* (for $\varepsilon = 0.1$) is better than the *GrGr* when the instance size is small (i.e. $m = \{8, 10\}$) regardless of their constraint levels. Moreover, *GrPTAS* also has good approximation ratios for $\varepsilon = 0.5$ for these instances as observed in 4.5(a) and 4.6(a). However, for the larger instances (i.e. $m = \{15, 20\}$) both the algorithms perform almost identical for all the constraint levels. In case of running time, *GrGr* is significantly faster than *GrPTAS* for all ε values as the load factor increases.

4.4 Scenario C: Performance with real life data

In this experiment we have used a real time dataset that contains mobility traces collected from taxis in San Francisco, USA. It contains GPS coordinates of approximately 500 taxis collected over 30 days. The dataset was provided by [47].

It is assumed that vehicles (taxis) from cabspotting database are equipped with processors. These processors are organized into a vehicular cloud around a fixed reference point on the San Francisco map. Given a radius (eg. 100 metres), the program will compute, based on the time stamp information from the database, the time intervals within the chosen radius of the fixed point. These time intervals become computing resource intervals of availability.

To generate the machine instances, we selected a reference point in San Francisco which is the location of City Hall. We simulate an application based on real life data where we assume that City Hall in cooperation with the nearby vehicles (i.e. in this case vehicles are are only the taxis which are available within 100m of city hall) can offload computing tasks for processing to the available vehicular resources instead of renting/outsourcing a computing infrastructure. The vehicle owners can be offered incentives such as monetary rewards or free parking service for the use of their resources so both parties benefit. Therefore, we computed the availability intervals according to the time interval in which taxis are within 100 meters radius of the City Hall as shown in Figure 4.9.

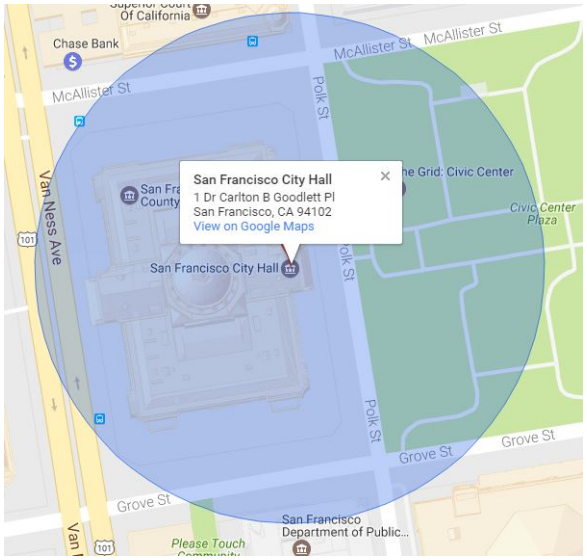


Figure 4.9: 100m radius around City Hall, San Francisco.

For this experiment, the simulation time window was chosen as one day (86400 seconds). We selected 10 slices of one-day duration each from the taxi mobility database collected within 100m radius of the City Hall. For the task requirements we have used traces from the DAS-2 system which is obtained from the Grid Workloads Archive (GWA) [9]. The *RunTime* field value in the trace file is considered as the *processing requirement* of a task that represents the duration (in seconds) required to complete that job. We selected 10 sets of task requirements from the DAS-2 system workload trace based on the running time of the jobs from 10 randomly chosen slices of one-day duration.

We assume that each vehicle has a specific processing speed and an usage cost per unit of time is associated with it based on its processing power. The cost-per-unit is chosen uniformly between 1 and 10. For each machine availability / task requirement pair, we assigned processor speeds uniformly at random to obtain a particular constraint level ranging from 100% (heavily loaded) to an instance of 10% (lightly loaded).

The instances extracted are extremely large, and so we have executed only the *GrGr* scheduler. From the previous experiments, it is evident that *GrGr* generates schedules with almost identical cost minimization performance ratios as the *GrPTAS* and has significantly better running time than the *GrPTAS* for the large size test instances. The goal of this

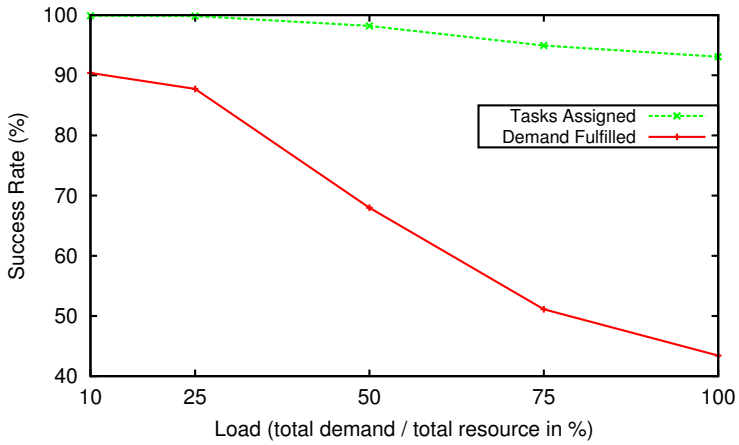
experiment is to observe the performance of *GrGr* algorithm in case of fulfilling the total demands in such real-life data.

Figure 4.10 shows the performance of the algorithm as a function of the system load (load factor). Fig. 4.10(a) depicts the success rate in terms of the fraction of demand fulfilled and the fraction of tasks completed as a function of the system load (load factor). Additionally, Fig. 4.10(b) shows the average duration of both tasks accepted and tasks rejected. Since we relax the constraint level in the graph from Fig. 4.10(b) by increasing the randomly assigned speed of the processors, the average duration of the accepted tasks as well as the rejected tasks decreases. In 4.10(b) we also represent a reference line (i.e. the blue line) corresponding to the average duration of the rejected tasks at 100% load. We thus notice that the average duration of the accepted tasks increases with the decrease in constraint level up to the 50% mark. Fig. 4.10(c) depicts the approximation ratio of the scheduler which is worse than in the case of randomly generated task requirements.

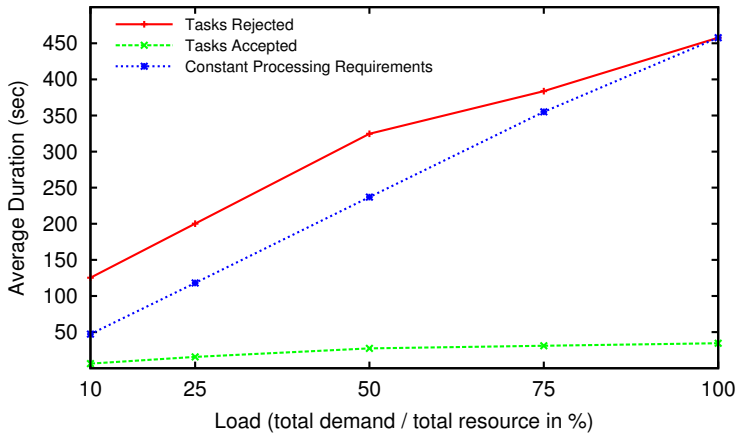
Observation Comparing figures 4.10(a) and 4.10(b) indicates that, *GrGr* was able to schedule most of the short tasks successfully whereas it failed to schedule some long tasks. Moreover, increasing the speed of the vehicles does not necessarily allow the vehicular cloud to process more tasks in some cases. We remark that, for any real life system, the tasks rejected by the vehicular cloud can be assigned to a fixed cloud infrastructure.

4.5 Concluding Remarks

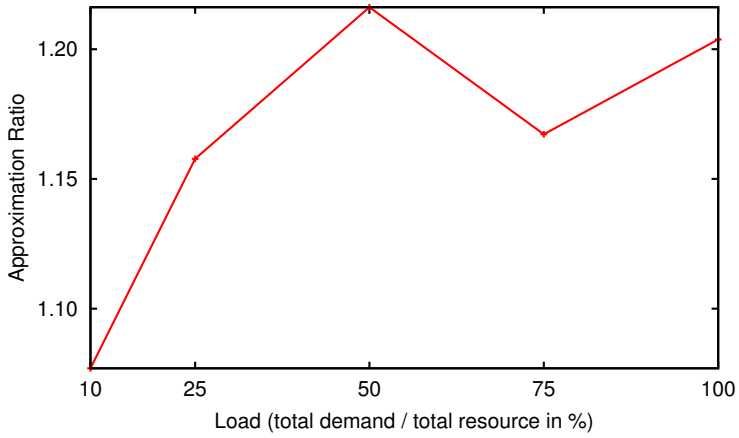
We observed that, the *GrGr* algorithm can produce a schedule with the minimum cost being almost similar to the *GrPTAS* algorithm in much lesser time. From the experimental results presented in sections 4.2.2 and 4.3.2 it is also visible that, for the *GrPTAS* algorithm, the running time reduces though the scheduling cost increases as the ε value increases. Moreover, from the results given in section 4.4 we see that, *GrGr* can also produce a schedule with good performance ratio which can be used with traditional cloud systems



(a) Success rate w.r.t the system load.



(b) Average duration w.r.t the system load.



(c) Approximation Ratios w.r.t the system load.

Figure 4.10: Performance of *GrGr* as a function of the system load on real life data.

side-by-side for handling the short tasks efficiently for real life applications. In summary, in terms of the cost minimization objective function *GrPTAS* outperforms *GrGr* for small size test instances and both the algorithms perform almost equal for large size instances. In terms of running time, *GrGr* shows significantly better performance than the *GrPTAS*.

Finally, it is mentioned earlier that the scheduling problem in consideration falls in the category of NP-complete problems. So, an algorithm which can solve a real life scheduling problem in shorter periods of time by producing a good result is in demand in today's highly competetive industrial environment. Analyzing above simulation results, we suggest that *GrGr* can be a good choice for task scheduling in such real life scenarios.

Chapter 5

Conclusion and Future works

5.1 Conclusion

Vehicular cloud systems have emerged from the motivation of taking the combined vehicular resources to the cloud for providing computational services. This pool of vehicular resources in a VC system implement the concept of fog computing, therefore plays a significant role in the Internet of Things.

One major difference between the conventional cloud and a VC is the mobiltiy of the vehicles which leads to a dynamic situation in terms of resource availability over time. In this thesis, we argued that task scheduling in vehicular clouds, unlike in any other distributed computing environment, requires the solution to a machine scheduling problem with special constraints.

We provide a first theoretical and empirical analysis of the scheduling problem and we present concrete measurements on the computational capacity of vehicular clouds. We proposed a greedy approach as a main procedure for solving the $n/U/VC$ problem that schedules n tasks by solving the $1/U/VC$ problem each time. Then we proposed two greedy heuristics for solving $1/U/VC$ problem based on a PTAS and a natural greedy procedure. We also propose an algorithm that provides guranteed accuracy for the $n/U/VC$ problem.

Our findings may not only prove useful to engineers considering an actual vehicular cloud deployment, but also for further research on algorithms for concrete resource scheduling problems in vehicular clouds.

5.2 Open Problems

In the future, it would be fruitful to investigate the following aspect of the resource assignment problem in vehicular clouds:

- Resource assignment strategy in an on-line environment with tasks becoming available in real time. Unlike the off-line algorithms, the whole input instance is not known to an on-line algorithm beforehand, that makes the scheduling scenario more realistic. Therefore, introducing natural heuristics for the task scheduling problem in vehicular clouds considering the on-line behavior of the tasks is an interesting open direction.

Bibliography

[1] Accelerating the world to sustainable energy. http://www.tesla.com/presskit/autopilot.

[2] Chip-maker driven by digital dream. http://www.driven.co.nz/news/chip-maker-driven-by-digital-dream/.

[3] Ford partners with microsoft azure to deliver cloud-based services and software updates. http://cloud-computing-today.com/tag/ford/.

[4] Nvidia-smart car super computers. http://www.theconnectedplanet.net/nvidia-smart-car-super-computers/.

[5] Fog computing and the internet of things: Extend the cloud to where the things are. White Paper, 2015.

[6] Sherin Abdelhamid. Towards provisioning vehicle-based information services. 2014.

[7] Sherin Abdelhamid, Robert Benkoczi, and Hossam S. Hassanein. Vehicular clouds - ubiquitous computing on wheels. Unpublished Book Chapter in Emergent Computation, edited by Andrew Adamatzky.

[8] Sherin Abdelhamid, Hossam S. Hassanein, and Glen Takahara. Vehicle as a resource (VaaR). *IEEE Network Magazine*, 29(1):12–17, 2015.

[9] Advanced School of Computing and Imaging (ASCI). GWA-T-1 DAS2. GWF format in Grid Workloads Archive, 2005.

[10] S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti, and H. Zedan. A comprehensive survey on vehicular ad hoc network. *Journal of network and computer applications*, 37:380–392, 2014.

[11] Samiur Arif, Stephan Olariu, Jin Wang, Gongjun Yan, Weiming Yang, and Ismail Khalil. Datacenter at the airport: Reasoning about time-dependent parking lot occupancy. *IEEE Transactions on Parallel and Distributed Systems*, 23(11):2067–2080, 2012.

[12] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[13] Flavio Bonomi. Connected vehicles, the internet of things, and fog computing. In *The Eighth ACM International Workshop on Vehicular Inter-Networking (VANET), Las Vegas, USA*, pages 13–15, 2011.

[14] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[15] Peter Brucker. *Scheduling Algorithms*, chapter Classification of Scheduling Problems, pages 1–10. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2007.

[16] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[17] János Csirik, Johannes Bartholomeus Gerardus Frenk, Martine Labbé, and Shuzhong Zhang. Heuristics for the 0-1 min-knapsack problem. *University of Szeged. Acta Cybernetica*, 10:15–20, 1991.

[18] R. Deng, R. Lu, C. Lai, and T. H. Luan. Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing. In *2015 IEEE International Conference on Communications (ICC)*, pages 3909–3914, June 2015.

[19] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang. Optimal workload allocation in fog-cloud computing towards balanced delay and power consumption. *IEEE Internet of Things Journal*, PP(99):1–1, 2016.

[20] A. Destounis, G. S. Paschos, and I. Koutsopoulos. Streaming big data meets backpressure in distributed network computation. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[21] Fangpeng Dong and Selim G Akl. Scheduling algorithms for grid computing: State of the art and open problems. Technical report, Queen's University, 2006.

[22] European Union Agency for Network and Information Security. Security framework for governmental clouds, February 2015.

[23] FedRAMP compliant systems. http://www.fedramp.gov/marketplace/compliant-systems/. retrieved on April 3, 2016.

[24] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[25] Georgii V Gens and Eugenii V Levner. Computational complexity of approximation algorithms for combinatorial problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 292–300. Springer, 1979.

[26] Teofilo Gonzalez, Oscar H Ibarra, and Sartaj Sahni. Bounds for lpt schedules on uniform processors. *SIAM journal on Computing*, 6(1):155–166, 1977.

[27] Teofilo F Gonzalez, Joseph Y-T Leung, and Michael Pinedo. Minimizing total completion time on uniform machines with deadline constraints. *ACM Transactions on Algorithms (TALG)*, 2(1):95–115, 2006.

[28] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[29] Ronald L Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[30] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

[31] Hannes Hartenstein and LP Laberteaux. A tutorial survey on vehicular ad hoc networks. *IEEE Communications magazine*, 46(6):164–171, 2008.

[32] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM (JACM)*, 23(2):317–327, 1976.

[33] Oscar H Ibarra and Chul E Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)*, 22(4):463–468, 1975.

[34] Mohammad Tauhidul Islam et al. *Approximation algorithms for minimum knapsack problem*. PhD thesis, Lethbridge, Alta.: University of Lethbridge, Dept. of Mathematics and Computer Science, c2009, 2009.

[35] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil. Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions. *IEEE Communications Surveys & Tutorials*, 13(4):584–616, 2011.

[36] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[37] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*, chapter The Multiple-Choice Knapsack Problem, pages 317–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[38] EL Lawler. Recent results in the theory of machine scheduling. In *Mathematical Programming The State of the Art*, pages 202–234. Springer, 1983.

[39] Eugene L Lawler, Jan Karel Lenstra, Alexander HG Rinnooy Kan, and David B Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522, 1993.

[40] Chung-Yee Lee, Lei Lei, and Michael Pinedo. Current trends in deterministic scheduling. *Annals of Operations Research*, 70:1–41, 1997.

[41] Peter Mell and Tim Grance. The NIST definition of cloud computing. Technical report, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

[42] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[43] S. Olariu, M. Eltoweissy, and M. Younis. Towards autonomous vehicular clouds. *ICST Transactions on Mobile Communications and Applications*, 11(7-9):1–11, 2011.

[44] Stephan Olariu, Tihomir Hristov, and Gongjun Yan. The next paradigm shift: from vehicular networks to vehicular clouds. *Mobile Ad Hoc Networking: Cutting Edge Directions, Second Edition*, pages 645–700, 2013.

[45] Stephan Olariu, Ismail Khalil, and Mahmoud Abuelela. Taking vanet to the clouds. *International Journal of Pervasive Computing and Communications*, 7(1):7–21, 2011.

[46] Christos H. Papadimitriou and Mihalis Yannakakis. Scheduling interval-ordered tasks. *SIAM Journal on Computing*, 8(3):405–409, 1979.

[47] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. CRAWDAD dataset epfl/mobility (v. 2009-02-24). Downloaded from `http://crawdad.org/epfl/mobility/20090224`, February 2009.

[48] Public Works and Government Services Canada. Request for information cloud computing solutions, December 2014. document number EN578-151297/B.

[49] Sartaj K Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM (JACM)*, 23(1):116–127, 1976.

[50] Günter Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15, 2000.

[51] SINTRONES. In-vehicle computing. `http://www.sintrones.com/products/invehiclecomputing.php`, 2015. Accessed: 18-03-2016.

[52] Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

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

[54] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.