Quality of Service Aware Reliable Task Scheduling in Vehicular Cloud Computing

Article *in* Mobile Networks and Applications · November 2015

DOI: 10.1007/s11036-015-0657-5

6 authors, including:

Tamal Adhikary
University of Dhaka
15 PUBLICATIONS **74** CITATIONS
SEE PROFILE

Amit Kumar Das
East West University (Bangladesh)
13 PUBLICATIONS **72** CITATIONS
SEE PROFILE

Md. Abdur Razzaque
University of Dhaka
84 PUBLICATIONS **566** CITATIONS
SEE PROFILE

Majed Alrubaian
King Saud University
46 PUBLICATIONS **118** CITATIONS
SEE PROFILE

Mobile Netw Appl
DOI 10.1007/s11036-015-0657-5

Quality of Service Aware Reliable Task Scheduling in Vehicular Cloud Computing

Tamal Adhikary[1] · Amit Kumar Das[1] · Md. Abdur Razzaque[1] · Ahmad Almogren[2] · Majed Alrubaian[2] · Mohammad Mehedi Hassan[2]

Abstract Vehicular Cloud Computing (VCC) facilitates real-time execution of many emerging user and intelligent transportation system (ITS) applications by exploiting under-utilized on-board computing resources available in nearby vehicles. These applications have heterogeneous time criticality, i.e., they demand different Quality-of-Service levels. In addition to that, mobility of the vehicles makes the problem of scheduling different application tasks on the vehicular computing resources a challenging one. In this article, we have formulated the task scheduling problem as a mixed integer linear program (MILP) optimization that increases the computation reliability even as reducing the job execution delay. Vehicular on-board units (OBUs), manufactured by different vendors, have different architecture and computing capabilities. We have exploited *MapReduce* computation model to address the problem of resource heterogeneity and to support computation parallelization. Performance of the proposed solution is evaluated in network simulator version 3 (ns-3) by running *MapReduce* applications in urban road environment and the results are compared with the state-of-the-art works. The results show that significant performance improvements in terms of reliability and job execution time can be achieved by the proposed task scheduling model.

Keywords Vehicular cloud · Task scheduling · MapReduce · MILP optimization · Quality-of-service

1 Introduction

Currently vehicles on the road carry a number of on-board computing devices having small-scale processing capabilities. A good number of research efforts on making the driving experience safer and smarter through the use of Vehicular Ad Hoc Network (VANET) applications, have recently been undertaken [1, 2]. However, as the vehicles are increasingly equipped with advanced sensors, a number of applications, ranging from inter vehicular driver healthcare and safety related applications to highway dynamic congestion management applications, have seen huge deployments using the data obtained from sensors. The huge data generated by the vehicular sensors need to be processed immediately, which gives birth to Vehicular Cloud Computing (VCC), wherein, vehicles collaborate to offer sensing, computing and storage services to the user applications [3–5]. More explicitly, VCC facilitates vehicular fleets on the roadways, streets, and parking lots to use their abundant and underutilized computing resources to provide widespread,

✉ Md. Abdur Razzaque
razzzaque@du.ac.bd

Tamal Adhikary
tamal.csedu@gmail.com

Amit Kumar Das
amit.csedu@gmail.com

Ahmad Almogren
ahalmogren@ksu.edu.sa

Majed Alrubaian
malrubaian.c@ksu.edu.sa

Mohammad Mehedi Hassan
mmhassan@ksu.edu.sa

[1] Green Networking Research Group, Department of Computer Science and Engineering, University of Dhaka, Dhaka-1000, Bangladesh

[2] College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

Published online: 12 November 2015

consistent, and reliable real-time services [3]. Olariu et al. [4] defined VCC as, "A group of largely autonomous vehicles whose corporate computing, sensing, communication and physical resources can be contributed and dynamically allocated to authorized users".

The difference between VCC and traditional cloud computing is that the VCC infrastructure and the resources are not fixed to a geographic location [12, 13]. The topology changes dynamically and VCC builds its platform opportunistically in a distributed manner, that is, the connectivity among the vehicles is neither reliable nor persistent. Vehicles have low computing capabilities and a VCC capitalizes spare and idle resources available on neighborhood vehicles. The services provided by a VCC have explicit lifetime, local validity, and most of the data-centric services are provider independent. In such an environment, the key research challenge is to select the neighborhood computing resources in such a manner that the application can be executed reliably within the delay deadline, that is, the Quality-of-Service (QoS) requirement are met.

In the literature, most of the research on VCC focuses on designing a framework for target applications in a centralized manner [6–9]. A resource provisioning mechanism using 'Bees Life' algorithm has been developed in Cloud Computing for Intelligent Transportation System (CCITS) [9], in which only load balancing among the vehicular workers is considered. Another time-constrained job-scheduling procedure has been presented in Cost Minimization Scheduling in Vehicular Cloud (CMSVC) [10], that minimizes the cost of execution of the tasks in a VCC. However, the main challenge in VCC is real-time service provisioning from analyzing an excessive volume of sensory data produced from a variety of on-board sensors such as: speedometer, engine, gas tank, camera, outside temperature sensor, and so on. None of the existing studies focus on enhancing the reliability of the execution of the jobs, for processing the sensory data in a time-critical manner. Again, in a centralized system, services cannot be provided in places when central communication and management infrastructure is not available. Therefore, it is crucial to develop methodologies to distribute the processing of data collected from sensors in a very short time limit.

In this article, we have designed an architecture of the infrastructureless VCC system, namely QoS Aware Reliable Task Scheduling (QARTS). We assume that a group of vehicles on the road having low relative velocities are clustered together [11]. Each cluster head (CH) exploits the *MapReduce* computing model for processing data in a fast and distributed manner. Each CH develops knowledge for all of it's worker member nodes in the reliability and timeliness domains, that is, the CH knows the expected time and reliability of completion if a given task of a job is assigned to a member node. Finally, we have formulated a mixed-integer linear programming (MILP) problem for optimal mapping of the application tasks to different worker nodes. To reduce the time complexity of MILP optimization, we have also developed a heuristic task scheduling algorithm. The major contributions of this study can be summarized as follows:

- An infrastructureless VCC framework using *MapReduce* model has been orchestrated.
- Quality-of-Service (QoS) and reliability driven optimal computation assignment problem for the application tasks to vehicular resources in VCC environment has been formulated as an MILP.
- Thereafter, a heuristic task-scheduling algorithm has been developed to achieve a feasible and near-optimal solution.
- The results of our extensive simulation experiments, carried out on network simulator version 3 (ns-3) [20], depict that significant performance improvements have been achieved by the proposed QARTS model compared to the state-of-the-art works.

The rest of the article is organized as follows. The state-of-the-art works on task scheduling in VCC have been described in Section 2. In Section 3, the assumptions and the system components of the VCC model along with their responsibilities have been demonstrated. The proposed solution of scheduling tasks in an optimal set of vehicular resources has been presented in Section 4. The performance evaluation of our proposed task-scheduling model has been demonstrated in Section 5, and the Section 6 concludes the article.

2 Related works

Different vehicular services can be combined, and new services can be designed by mapping, encapsulation, and aggregation of services through the VCC model. Cloud services in vehicular environment can be classified into following three types:

- *Vehicles using cloud:* Vehicles obtain cloud services from traditional cloud computing datacenters by uploading jobs to the cloud.
- *Vehicular cloud:* Vehicles experience cloud services from other vehicles, that is, vehicles volunteer or get paid for renting their underutilized resources.
- *Hybrid cloud:* Traditional cloud and vehicular cloud jointly provide infrastructure for computation.

Authors in [8, 14], and [15] opted for traditional cloud computing model with road side unit (RSU) or a base station (BS) as a central coordinator. This model is criticized by additional cost for infrastructure establishment. A hybrid VCC architecture has been considered in [6] and [16] that

uses both the traditional cloud and temporary VCC. However, they did not specify where and how to make this scheduling decision. In this study, we have focused on vehicular cloud [7, 9, 10], where, vehicles form temporary service clusters and obtain cloud services from cluster workers. The cluster heads (CH) take the decision of where, when, and how to execute the jobs in a distributed manner for the running vehicles.

The authors in CCITS [9] select servers for task scheduling from datacenters and vehicles with same priorities. It considers the capacity of the servers to execute tasks with minimum time. It uses 'Bees' Life' algorithm that explores servers repeatedly through a fitness function and chooses one that produces the best fitness value. However, for reliable computation in a mobile environment, it does not consider the connectivity levels among the vehicles, which is an important consideration.

The authors in CMSVC [10] use binary integer programming model that takes into account the current available VCs, cost information and characteristics of applications. They consider that the tasks are executable in parallel and once assigned to an instance, cannot be reassigned to another one. However, in practical, a worker vehicle may leave the VCC service at any time and duplicate workers might be assigned to take over the responsibility.

None of the existing studies focus on increasing the reliability on executing tasks in a VCC. Because of mobility of the vehicles, the topology of connectivity changes very rapidly. Hence, reliability in computation is a very significant threat in such an environment. Again, for computing and analyzing huge amount of sensor data, a distributed computing framework is required. These observations lead us to design a job execution framework for infrastructureless VCC environment using *Hadoop MapReduce* computation model.

3 VCC model and assumptions

We consider a vehicular cloud computing environment where sensing and computing services are provided by on board units (OBUs) on nearby vehicles. An OBU has low-scale computation and storage capability that the vehicle owner can rent or share with others. In this study, we allow a group of vehicles, moving in the same direction with very low relative speed, to form a cluster. The CH manages the computation of jobs submitted by any of its member nodes using *MapReduce* distributed computation model [19, 25].

3.1 Vehicular cluster formation

The vehicles in the road move with high velocity, which results in rapid topology changes, unstable communications, and high overhead for exchanging new topology information to all the vehicles. To improve the data routing among the vehicles and to increase the QoS offered by a VCC, vehicles moving in the same direction are clustered. The responsibility of a CH is to manage cluster membership and media access among the member workers, traffic control, decomposition of a job into tasks, distribute tasks among the cluster workers, collect results from them and send the result back to the requester.

A good number of cluster formation techniques for VANET environment are studied in the literature [11, 23, 24]. We have followed the clustering algorithm described in [11] with very simple modification. In selecting the CH, we have considered the amount of available processing resources at the vehicles in addition to throughput performances as in [11], which is measured by exploiting channel condition, vehicle movement pattern and MAC protocol performance. The vehicle having the highest value for the product of normalized throughput and resource availability has been chosen as the CH. This product metric helps to choose moderate to high performing nodes as CHs and excludes the others poor in resource or throughput performance. Alternative approaches can also be exploited for further optimizing the clustering performance, which is out of the scope of this work.

3.2 Job execution model

Inside each cluster, the processing of tasks is performed using the *MapReduce* big data processing model. *MapReduce* is a distributed computing framework for processing a large volume of raw data. It was modeled after *Google's* article on *MapReduce* and *Hadoop* is an open source implementation of *MapReduce*. The *MapReduce* model simplifies computational parallelization, work distribution, and machine reliability helping VCC application developers focus on specific business goals.

The *MapReduce* follows a batch processing, shared nothing model for removing the parallel execution interdependencies. Programmers define the *Map* and *Reduce* functions. The *Map* function performs pre-processing on input tasks and generates intermediate result of (key, value) pairs as follows,

$$Input\_task(x_i) \xrightarrow{\text{Map}} ilist(key, value). \quad (1)$$

The *Map* function categorizes the input data and makes it ready to be processed by the *Reduce* function. The *Reduce* function works on the intermediate list, $ilist$ of (key, value) pairs, and generates the final output as follows,

$$ilist(key, value) \xrightarrow{Reduce} Output(y_i). \quad (2)$$

The *MapReduce* framework for a VCC is shown in Fig. 1. The CH performs the role of the *MapReduce* master. The member nodes send their job execution request to the CH, which partitions the job into a number of *Map* tasks. The *Map* tasks are atomic, that is, either a task execution is fully completed or failed. The *Map* tasks are scheduled in a set of *Worker* vehicular resources. The results of these *Map* tasks are then passed to another set of *Worker* resources to perform *Reduce* tasks on the result of *Map* tasks. The results of the *Reduce* tasks are then passed to the CH, which finally sends the results to the requesting vehicle. The *MapReduce* design principle described in [25] has been used in this study for improving the resource utilization inside each of the workers.

In a cluster, the resources available at each of the member vehicles are notified to the CH. Whenever vehicles get connected to a CH, they advertise their sensing and computing resources to the CH. The connected vehicles also periodically send their current resource availability information to the CH. The virtualization process is performed at the CH and it keeps track of all the available resources the workers currently possess.

Figure 2 shows our VCC architecture model and the system components of the CH and its worker members. Users can run applications requiring higher computation and memory capacity. Applications that cannot be run on local resources are sent to the CH which exploits VCC facilities for the required computations. The details of optimal task scheduling performed by the CH, is presented in Section 4.

In the CH, the *Task Scheduler* distributes the tasks to the selected workers according to the capacity of each worker. The capacity of a worker is defined by the number of slots

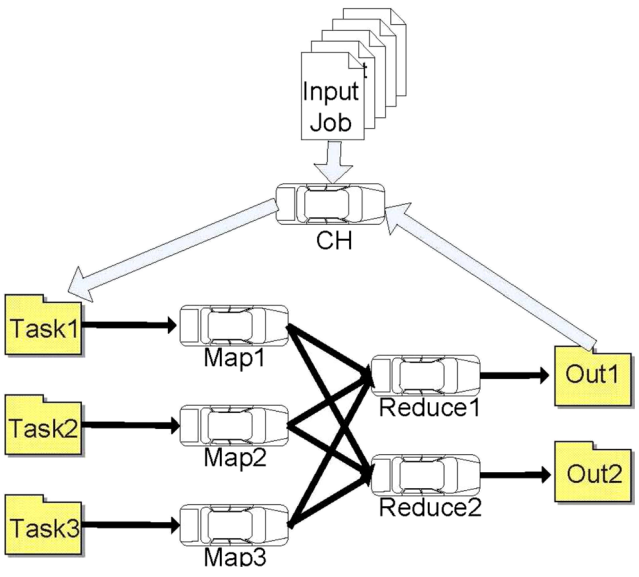


Fig. 1 MapReduce Framework for VCC

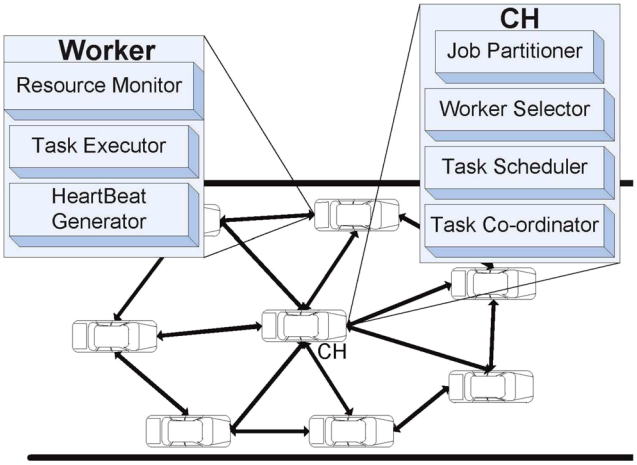


Fig. 2 VCC System Model Using MapReduce

it contains. A slot is a simple abstraction of the physical resource available on the vehicles. The number of *Map* and *Reduce* slots in each vehicle is fixed and the slot/core ratio is specified by the system designer. Each slot can run only a single task at a given time. A slot overcomes the problem of machine heterogeneity because vehicles with different capacities, advertise different number of free slots they contain. The *Task Coordinator* at CH periodically checks the connectivity and availability of each worker. It reschedules tasks assigned to workers that have gone outside of the cluster, increasing the reliability of the system.

Each worker node monitors available slots it has and periodically informs the CH through a *Heartbeat* message in the form of the follwoing tuple,

$$< ID,\ loc(x, y),\ speed,\ acceleration,\ free\_slot > .$$

The *Heartbeat Generator* periodically generates the *Heartbeat* frame containing the number of available slots, the current speed, location, and acceleration of the vehicles and sends the *Heartbeat* frame to the CH. *Resource Monitor* checks the availability of slots, the task queue for each slot, the load of each slot, and the load of the processor to determine the number of available slots. The *Task Executor* loads tasks from the input queue of each slot and keeps track of the state of every task.

The proposed VCC architecture allows vehicles with heterogeneous computing resources to collaborate for supporting a number of applications, including a) network and data processing as a service [6]; b) intelligent transportation system by exploiting group mobility [7, 9]; c) crowdsourcing through collection and processing of large volume of sensor data from neighboring vehicles, which could be applied for dynamic traffic signal optimization, evacuation management, vehicular on-road safety assurance applications, congestion mitigation, parking management, and so on.

Table 1 Notations used in this paper

Notation	Description
$\mathcal{J}$	A job request
$\mathcal{M}$	Set of all member workers in a cluster
$\mathcal{S}_m$	Set of slots available in worker $m \in \mathcal{M}$
$\mathcal{B}$	Bit rate for wireless channels (bps)
D_{ϕ}^{jms}	Delay for transmitting a *Map* task $j \in \mathcal{J}$ at worker $m \in \mathcal{M}$ in slot $s \in \mathcal{S}_m$
D_{P1}^{jms}	Delay for processing a *Map* task $j \in \mathcal{J}$ at worker $m \in \mathcal{M}$ in slot $s \in \mathcal{S}_m$
D_{ψ}^{jms}	Delay for transmitting a *Reduce* task $j \in \mathcal{J}$ at worker $m \in \mathcal{M}$ in slot $s \in \mathcal{S}_m$
D_{P2}^{jms}	Delay for processing a *Reduce* task $j \in \mathcal{J}$ at worker $m \in \mathcal{M}$ in slot $s \in \mathcal{S}_m$
D_{R}^{jms}	Delay for collecting results of a *Reduce* task $j \in \mathcal{J}$ at worker $m \in \mathcal{M}$ in slot $s \in \mathcal{S}_m$
ν_m	Instruction execution rate of a worker $m \in \mathcal{M}$
δ_m	Contact rate of a worker $m \in \mathcal{M}$ with the CH
t_m^j	Execution delay of task $j \in \mathcal{J}$ in a slot of worker $m \in \mathcal{M}$
σ_m	Inter contact period between a worker $m \in \mathcal{M}$ and CH
$p_m(t)$	Contact probability of worker $m \in \mathcal{M}$ with the CH
p_m^j	Success probability of task $j \in \mathcal{J}$ if executed in worker $m \in \mathcal{M}$
α	Relative priority factor between response time and success probability

[17]; d) photo surveillance and homeland security application [18]; e) driver healthcare applications and f) translation of statements in different languages. The notations used in this article are summarized in Table 1.

4 Optimal task scheduling

The optimal scheduling of the tasks in a job refers to the problem of selecting a group of workers that are connected with high reliability and provide low response time. A worker is said to be reliably connected with the CH if it does not lose the connectivity before sending back the task execution result within the deadline. The response time is defined by the time interval in between the time at which a task is submitted to the worker and the time at which the result of the task is collected back. Workers, that have higher number of *Map* and *Reduce* slots, offer reduced communication latency for sharing intermediary results.[1] Again workers, that have faster processing units, offer a low response time. Considering the above facts, in this section, we develop an objective function that selects the optimal set of workers for task execution and set the constraints that the objective function follows. We first present the methodologies for calculating the metrics *response time* and *success probability* that are exploited by the objective function.

[1]Because in the case, the *Map* and *Reduce* slots reside in the same worker with higher probability.

4.1 Calculation of response time

The response time of executing a job is composed of times to a) disseminate *Map* tasks, b) execute *Map* tasks, c) send results of *Map* tasks to *Reduce* tasks, d) execute *Reduce* tasks, and e) collect results. The communication and computation of different subtasks on multiple cluster members are independent of each other. For example, processing of a *Reduce* task starts as soon as it gets results from a *Map* task; it does not wait for all the other *Map* tasks to complete. Therefore, the sum of maximum times required for transmission and processing of subtasks gives the worst case value of job response time. Let $\mathcal{M}$ be the set of available workers and $\mathcal{S}_m$ be the set of slots available in worker $m \in \mathcal{M}$. The binary variable A_{jms} has value 1, when a task $j \in \mathcal{J}$ is scheduled to be executed in slot $s \in \mathcal{S}_m$ of worker $m \in \mathcal{M}$, and 0, otherwise.

The CH partitions a submitted job $\mathcal{J}$ into $j\prime$ tasks to be executed by *Map* slots and then the results are passed to $j - j\prime$ *Reduce* slots. The functions, $f'()$ and $f''()$ provide instructions for performing the operations on *Map* and *Reduce* tasks, respectively. Therefore, the set of all *Map* and *Reduce* tasks, can be represented by

$$\mathcal{J} = \{x_1, x_2, ..., x_j \mid (\forall j > j\prime, \exists j'' \leq j\prime)\, x_j = f'(x_{j''})\}. \tag{3}$$

Appropriate j number of slots from available workers are then selected to schedule the *Map* and *Reduce* tasks. Let Y be the set of output components of the *Map* and *Reduce* tasks and it is represented by

$$Y = \{y_1, y_2, ..., y_j \mid (\forall j \in \mathcal{J})\, y_j = f'(x_j \mid j \leq j\prime) \vee f''(x_j \mid j > j\prime)\}. \tag{4}$$

The execution time of $f'(x_j)$ and $f''(x_j)$ are different depending on types of applications. Each worker sends its connectivity list containing its neighboring workers $m \in \mathcal{M}$ to the CH. From the connectivity lists, received from the workers, the CH creates a two dimensional connectivity matrix C. Each entry C_{mn} is determined as follows:

$$C_{mn} = \begin{cases} 0 & \text{if } m = n, \\ 1 & \text{if } m \text{ and } n \text{ are neighbors}, \\ 2 & \text{if } m \text{ and } n \text{ are connected via the CH.} \end{cases} \tag{5}$$

The first condition of Eq. 5 states that the *Map* and *Reduce* tasks can be computed on different slots of a single

worker. Our optimal task scheduling approach tries to minimize the C_{mn} values for the assignment of different *Reduce* tasks to available slots on workers. For simplicity, we have considered that all the channels associated with the links among the workers and the CH have the same bit rate. For calculating the dissemination and execution times of *Map* and *Reduce* tasks, a task that takes the longest time to be transmitted and executed is considered as the total execution time, since the transmission and execution in different workers go in parallel. Considering $\mathcal{B}$ as the bit rate for wireless channels (in bps), the delay associated with a worker $m \in \mathcal{M}$ for receiving *Map* tasks, can be denoted as,

$$D_{\phi}^{jms} = \max_{\forall j \in \mathcal{J} | j \leq j'} \{A_{jms} \cdot \frac{x_j}{\mathcal{B}}\}. \tag{6}$$

The execution delay of a task x_j is different for different workers as the on-board vehicle units might have diverse computation capabilities. The time elapsed to process x_j for $\forall j \in \mathcal{J} \mid j \leq j'$ by a worker $m \in \mathcal{M}$ for processing the *Map* tasks is given by,

$$D_{P1}^{jms} = \max_{\forall j \in \mathcal{J} | j \leq j'} \{A_{jms} \cdot \frac{f'(x_j)}{\nu_m}\}, \tag{7}$$

where, ν_m represents the number of instructions executed in unit time interval by a slot of machine m. After completion of the *Map* tasks in any of the workers $m \in \mathcal{M}$, the delay associated with transmitting the result to $|\mathcal{J}| - j'$ *Reduce* tasks running in any of the workers $n \in \mathcal{M}$, can be computed as

$$D_{\psi}^{jms} = \max_{\forall j \in \mathcal{J} | j \leq j'} \{A_{jms} \cdot (|\mathcal{J}| - j') \cdot C_{mn} \cdot \frac{y_j}{\mathcal{B}}\}. \tag{8}$$

The time elapsed to process x_j for $\forall j \in \mathcal{J} \mid j > j'$ by a worker $m \in \mathcal{M}$ for processing the *Reduce* tasks can be expressed as

$$D_{P2}^{jms} = \max_{\forall j \in \mathcal{J} | j > j'} \{A_{jms} \cdot \frac{f''(x_j)}{\nu_m}\}. \tag{9}$$

Finally, delay for transmitting the results of *Reduce* tasks to the CH can be given by

$$D_{R}^{jms} = \max_{\forall j \in \mathcal{J} | j > j'} \{A_{jms} \cdot \frac{y_j}{\mathcal{B}}\}. \tag{10}$$

Summing up all these delays calculated in Eqs. 6–10, the overall response time for a job is computed as

$$D_T^{\mathcal{J}} = D_{\phi}^{jms} + D_{P1}^{jms} + D_{\psi}^{jms} + D_{P2}^{jms} + D_{R}^{jms}. \tag{11}$$

In contrast to that, if the tasks of the job were computed locally, the required time could be expressed as

$$T_o = \frac{\sum_{\forall j \in \mathcal{J} | j \leq j'} f'(x_j) + \sum_{\forall j \in \mathcal{J} | j > j'} f''(x_j)}{\nu_o \times \mathcal{S}_o}, \tag{12}$$

where, ν_o is the instruction execution rate of a single slot of the job requester and $\mathcal{S}_o$ is the number of slots available in it.

4.2 Calculation of success probability

The vehicular workers that have higher probability of contact with the CH have higher reliability for computation of tasks before deadline. Because the position and velocity of workers are highly dynamic, reliability is an important factor to select workers as it helps avoid work failure and time penalties. Rework is needed in case a worker leaves the cluster and does not return before the deadline. Deadline for a task, $\mathcal{T}$, will expire, if the job requester leaves the cluster or the urgency period of the application expires.

A worker server that has higher probability of completing a task before $\mathcal{T}$ is chosen for computation service. The probability is calculated using the contact rate of a worker vehicle with the CH. The contact rate of a worker $m \in \mathcal{M}$ is defined as $\delta_m = 1/E[\sigma_m]$, where σ_m represents the inter contact period between the CH and the worker $m \in \mathcal{M}$.

Contact between a worker m and the CH is Poisson distributed with the contact rate δ_m. The expectation of success ratio of a job can be maximized by maximizing the success probability of each individual task. The time needed by a worker $m \in \mathcal{M}$ to execute a task x_j is given by $t_m^j = f(x_j)/\nu_m$, where $f()$ is either $f'()$ or $f''()$. Therefore, for successful completion of the task before deadline $\mathcal{T}$, demands the worker m to contact CH before the time $\mathcal{T} - t_m^j$.

The contact probability $p_m(t)$ of a worker $m \in \mathcal{M}$ in the interval $0 \sim t$ for any value of t in the range $(0 \leq t \leq \mathcal{T} - t_m^j)$ when the worker m first meets the CH, can be expressed as

$$\begin{aligned} p_m(t) &= \int_0^{\mathcal{T}-t-t_m^j} \delta_m \cdot \mathrm{e}^{-\delta_m y}\, dy \\ &= 1 - \mathrm{e}^{-\delta_m(\mathcal{T}-t-t_m^j)}. \end{aligned} \tag{13}$$

The success probability p_m^j of computing a task $j \in \mathcal{J}$ on worker $m \in \mathcal{M}$, is the cumulative distribution function (CDF) of the contact probability $p_m(t)$ for all values of t in the interval $0 \sim \mathcal{T} - t_m^j$, calculated as follows

$$\begin{aligned} p_m^j &= \zeta_m \times \int_0^{\mathcal{T}-t_m^j} p_m(t) \cdot \delta_m \cdot \mathrm{e}^{-\delta_m t}\, dt \\ &= \zeta_m \times \{1 - (\delta_m(\mathcal{T} - t_m^j) + 1)\mathrm{e}^{-\delta_m(\mathcal{T}-t_m^j)}\}, \end{aligned} \tag{14}$$

where, the binary variable ζ_m represents availability of slots in a worker $m \in \mathcal{M}$; $\zeta_m = 1$ if the worker m has at least one free slot for computation and $\zeta_m = 0$ otherwise. Now, to increase the success probability of the job, we need to assign tasks in such a way that the total success probability for all the tasks is maximized. The integrated success probability of all the tasks in job $\mathcal{J}$ can be expressed as

$$S(\mathcal{J}) = \prod_{\forall j \in \mathcal{J}} p_m^j \mid p_m^j > 0. \tag{15}$$

4.3 Optimization problem formulation

The problem of selecting an optimal set of vehicular workers can now be expressed as a MILP optimization problem. The target is to increase the success probability $S(\mathcal{J})$ for all the tasks in job $\mathcal{J}$ even as minimizing the total service time, $D_\tau^{\mathcal{J}}$. We formulate the objective function as follows:

$$\max Z = \alpha \times S(\mathcal{J}) - (1-\alpha) \times \frac{D_T^{\mathcal{J}}}{T_o}, \tag{16}$$

where, α is a weight parameter specified by the job requester for the application.

The parameter α gives the relative priority between the job completion time and success probability. Some applications might be delay-tolerant but require higher probability for being processed successfully. Some other real-time applications might emphasize on job completion time rather than success probability. When submitting their jobs to the CH, the requesters specify the impact of response time ($\mathcal{I}_r$) and success probability ($\mathcal{I}_s$) on executing their jobs in a scale of 10. The value of α is calculated from these two parameters as

$$\alpha = \frac{\mathcal{I}_s}{\mathcal{I}_r + \mathcal{I}_s}. \tag{17}$$

4.4 Constraints

- **Assignment constraint:**

$$A_{jms} \in \{0,1\}, \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}, \forall s \in \mathcal{S}_m \tag{18}$$

$$\sum_{\forall m \in \mathcal{M}} \sum_{\forall s \in \mathcal{S}_m} A_{jms} = 1, \quad \forall j \in \mathcal{J} \tag{19}$$

  At most one task can be allocated to a particular slot of a worker at a given time, and some worker slots may remain free. A task is executed only in a single slot and no task mirroring is allowed.
- **Job integrity constraint:** Each of the tasks x_j of a job $\mathcal{J}$ must be allocated to any of the slots of workers. No tasks can be lost or kept unallocated.

$$\sum_{\forall j \in \mathcal{J}} \sum_{\forall m \in \mathcal{M}} \sum_{\forall s \in \mathcal{S}_m} A_{jms} = |\mathcal{J}| \tag{20}$$

- **Capacity constraint:** The number of tasks allocated to a worker should not overflow its capacity.

$$\sum_{\forall j \in \mathcal{J}} \sum_{\forall s \in \mathcal{S}_m} A_{jms} \leq |\mathcal{S}_m|, \quad \forall m \in \mathcal{M} \tag{21}$$

- **Slot availability constraint:** The total number of available slots in all the workers should be higher than or equal to the number of tasks in a job; otherwise, the execution of the job is not possible.

$$|\mathcal{J}| \leq \sum_{\forall m \in \mathcal{M}} |\mathcal{S}_m| \tag{22}$$

- **QoS constraint:**

$$D_T^{\mathcal{J}} < T_o \tag{23}$$

$$D_T^{\mathcal{J}} \leq \mathcal{T} \tag{24}$$

The execution time of a job in VCC should be less than the time required to execute the task locally in the job initiator. Sometimes, the time taken by the workers might be more than the job initiator because of poor resource availability, excessive communication delay, job partitioning complexity, *MapReduce* dependency among the workers, and so on. The constraint in Eq. 24 states that all the tasks should be executed within the deadline $\mathcal{T}$.

4.5 Heuristic task scheduling

The optimal task scheduling approach, presented in the previous section, takes ample time to converge to an optimal solution because of the curse of dimensionality. However, in vehicular environment, the execution resource assignment decisions need to be taken at real-time, causing minimum waiting time for the application job requests. Furthermore, solving an MILP problem demands higher computing facility, which is quite scarce in vehicular on-board units. In this section, we develop a heuristic solution for the task scheduling problem that reduces the complexity significantly and achieves near-optimal solution.

The heuristic property that we have exploited in this solution is that workers that satisfy a minimum execution success threshold (p^j) for a given task $j \in \mathcal{J}$ and require the least task execution time, will maximize Z, that is, the following inequality must hold for all workers:

$$\forall j \in \mathcal{J} | p_m^j \geq p^j \{p^j \cdot \frac{1}{t_m^j}\} \geq \forall j \in \mathcal{J} | p_m^j < p^j \{p_m^j \cdot \frac{1}{t_m^j}\}, \quad \forall m \in \mathcal{M}. \tag{25}$$

The minimum execution success threshold p^j for each task $j \in \mathcal{J}$ is calculated by taking the mean of the successful task completion probabilities at all workers and scaling it with the relative priority factor α as follows:

$$p^j = \begin{cases} a\bar{p^j} \times 2\alpha & \text{if } \bar{p^j} \times 2\alpha \leq \max\limits_{\forall m \in \mathcal{M}} (p_m^j), \\ \max\limits_{\forall m \in \mathcal{M}} (p_m^j) & \text{otherwise,} \end{cases} \tag{26}$$

where,

$$\bar{p^j} = \frac{\sum_{\forall m \in \mathcal{M}} p_m^j}{|\mathcal{M}|}.$$

Note that the $\bar{p^j} \times 2\alpha$ value scales up (if $\alpha > 0.5$) or scales down (if $\alpha < 0.5$) the minimum success threshold p^j from mean $\bar{p^j}$ for all workers or keeps it unchanged (when

$\alpha = 0.5$). In the worst case, when none of the workers satisfy the threshold, the p^j value is bounded by maximum p_m^j $\forall m \in \mathcal{M}$. This dynamic control of the threshold facilitates our QARTS system to select more suitable candidate workers for executing tasks of a job.

Algorithm 1 summarizes the steps of heuristic task scheduling approach using *MapReduce*. At first, for each task $j \in \mathcal{J}$, a set of workers (M_p) are selected based on their success probabilities (p_m^j) (line no. 9). In the case, the M_p produces an empty set, that is, no worker satisfies the minimum threshold of successful task execution probability, then the search for slots stops. Otherwise, the task processing delays (t_m^j) in the candidate slots on the vehicular workers are calculated. Similarly, the delays for data transmission d_t^j from a candidate slot to the other selected slot(s) are also calculated (line no. 17). Then, the candidate slot that requires minimum time for processing and data transmission has been selected and added to the set of selected slots (S_Ω) (line no. 20-22). Finally, we produce A_{jms} that gives the mapping of each task $j \in \mathcal{J}$ scheduled to be executed in a slot $s \in \mathcal{S}_m$ of worker $m \in \mathcal{M}$.

Algorithm 1 Heuristic Task Scheduling Algorithm

INPUT: *$\mathcal{J}$: Set of tasks, $\mathcal{M}$: Set of vehicular workers, $\mathcal{S}_m$: Set of slots in the worker $m \in \mathcal{M}$, α: Relative priority between response time and success probability*

OUTPUT: *A_{jms}: Scheduling a task $j \in \mathcal{J}$ on worker $m \in \mathcal{M}$ in slot $s \in \mathcal{S}_m$*

1. $S_\Omega \leftarrow \Phi$
2. $A_{jms} = 0 \mid \forall j \in \mathcal{J}, \forall m \in \mathcal{M}, \forall s \in \mathcal{S}_m$
3. **for** each task $j \in \mathcal{J}$ **do**
4. Calculate α using Eq. 17
5. **for** each worker $m \in \mathcal{M}$ **do**
6. Calculate p_m^j using Eq. 14
7. **end for**
8. Calculate p^j using Eq. 26
9. $M_p \leftarrow \{m_1, \cdots, m_k \mid \forall m \in \mathcal{M},\ p_m^j \geq p^j\}$
10. **if** $M_p = \Phi$ **then**
11. Exit
12. **end if**
13. **for** each slot $s \in \mathcal{S}_m \mid m \in M_p$ **do**
14. $t_m^j \leftarrow \frac{f(x_j)}{\nu_m}$
15. $d_t^j \leftarrow 0$
16. **for** each slot ω in S_Ω **do**
17. $d_t^j \leftarrow d_t^j + \frac{x_j}{\mathcal{B}} \times C_{mn} \mid \omega \in \mathcal{S}_n$
18. **end for**
19. **end for**
20. $s_j \leftarrow \underset{s}{\operatorname{argmin}}\{t_m^j + d_t^j\}$
21. $\mathcal{S}_m \leftarrow \mathcal{S}_m \backslash s_j$
22. $S_\Omega \leftarrow S_\Omega \cup s_j$
23. $A_{jms} = 1 \mid s = s_j \ \&\ s_j \in \mathcal{S}_m$
24. **end for**

The complexity of Algorithm 1 is quite straightforward to follow. The statements 3 $\sim$ 24 are enclosed in a loop that iterates $|\mathcal{J}|$ times, and 5 $\sim$ 7, 8 and 9 have the same complexity of $O(|\mathcal{M}|)$. The statements 13 $\sim$ 19 iterate $|\mathcal{S}_m|$ times in the worst case and inside those statements 16 $\sim$ 18 iterate $|\mathcal{S}_\Omega|$ times, where $\mathcal{S}_\Omega \subseteq \mathcal{S}_m$. The rest of the statements have constant unit time complexities. Therefore, the overall computational complexity of the algorithm is $O(|\mathcal{J}|.|\mathcal{M}| + |\mathcal{J}|.|\mathcal{S}_m|^2) \approx O(|\mathcal{J}|.|\mathcal{S}_m|^2)$.

5 Performance evaluation

We have evaluated our proposed task scheduling model in ns-3 [20] combined with *Hadoop MapReduce* simulator MRPerf [21] and the simulator of Urban Mobility (SUMO), which is a vehicle mobility simulator [22]. The results given here are based on the system performance, which we have found from the comparative study among QARTS, CMSVC [10], and CCITS [9] models. For generating the data and traffic for *Map* and *Reduce* tasks, we have used MRPerf, which is an open-source *MapReduce* simulator for designing decisions in *MapReduce* setups based on *Hadoop*. For tracing vehicular mobility and vehicles' behavior in the urban environment, we have used SUMO, which is also an open-source microscopic traffic road simulation package designed to handle large road networks. The MRPerf and SUMO simulators are synchronized and run with the ns-3. To evaluate the influence of the different parameters on the optimization function, we have used NEOS optimization tool [26] to solve the MILP problem.

5.1 Simulation environment setup

First, we run MRPerf from the ns-3, which simulates *MapReduce* in a virtual environment. Then, we measure the data volume processed, read and transmitted by each of the *Map* and *Reduce* tasks from MRPerf. Subsequently, we generate a vehicular mobility trace file for preferred speeds and priorities and import it into the ns-3 tool. We have considered an environment containing 20-50 vehicles per kilometer road area with speed 36-90 km/h. Each vehicle is considered to have a transmission range of 100m. An urban environment of square size with 500m length on each side is created into SUMO, as shown in Fig. 3. There are also cross roads at 250m on each side of the square.

For data transmission among the vehicles, we have used Wireless Access in Vehicular Environment (WAVE) (IEEE 802.11p) standard, which has 7 channels each with 6 Mbps data rate. The size of each packet is 1024 bytes. Each vehicle is considered to have 2-6 *Map* or *Reduce* or a combination of both type of slots. Each *Map* slot is configured with 1500 MIPS and each *Reduce* slot with 2000 MIPS. Each worker contains 1GB or 2GB RAM and 40-100 GB of secondary storage. In such an environment, we have scheduled 2-8 applications per minute, each consisting of 10-20 tasks. Each of the tasks requires different execution periods based

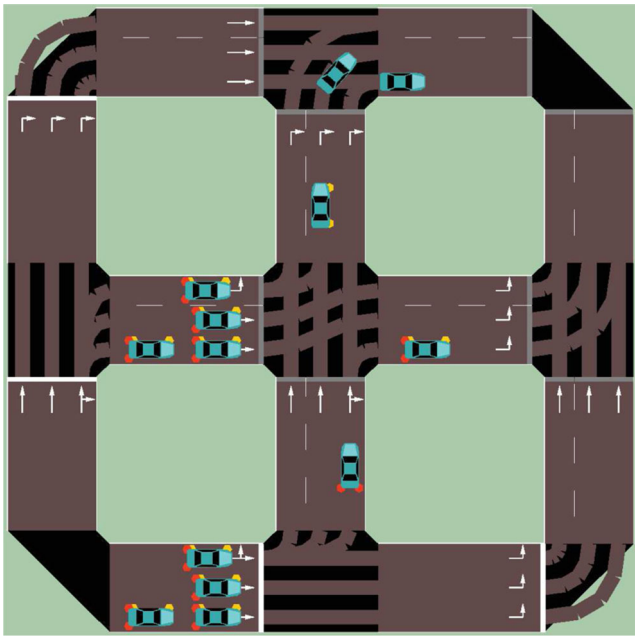


Fig. 3 Road environment in SUMO

on the number of instructions and inputs. The values of $\mathcal{I}_s$ and $\mathcal{I}_r$ are set to 6 and 4, respectively so as to accept higher impact of success probability of job execution compared to its time criticality; this consideration is practical for many VCC applications including safe transportation, traffic congestion management, etc. The aforementioned values are fed into the model as random variables with Normal Gaussian distribution. The heart-beat message interval is kept at 80ms and 40ms in implementing two versions of the proposed system, namely QARTS-1 and QARTS-2, respectively. Simulation is run for 1000 seconds and the results of 20 individual simulation runs are averaged to reduce the influence of random variability in the inputs.

5.2 Performance metrics

The comparative performances of the studied systems have been evaluated based on the following metrics.

- *Average execution time of jobs:* The execution time of an individual job is measured as the time duration between its submission to the time of obtaning results back; the average is taken for all jobs to quantify the execution time of jobs required by a studied scheduling mechanism.
- *Successful job execution (%):* A job execution is considered successful when the CH gets back the results from all the workers scheduled for task executions. Then, successful job execution percentage is measured using the number of successfully completed jobs and the total number of submitted jobs during the whole simulation period.
- *System throughput:* System throughput of a job execution model is measured by the number of jobs executed in it per unit time.
- *Task execution overhead:* Task execution overhead, as depicted in Eq. 27, quantifies percentage of total response time for a job that is wasted for data communication, task scheduling, re-scheduling, and reprocessing caused by failures other than actual execution times D_{P1}^{jms} and D_{P2}^{jms} at *Map* and *Reduce* slots, respectively.

$$Overhead = (1 - \frac{D_{P1}^{jms} + D_{P2}^{jms}}{D_T^{\mathcal{J}}}) \times 100. \tag{27}$$

The individual overhead percentages for the different jobs are then averaged for all the jobs executed during the simulation period to quantify the average task execution overhead of a scheduling approach.

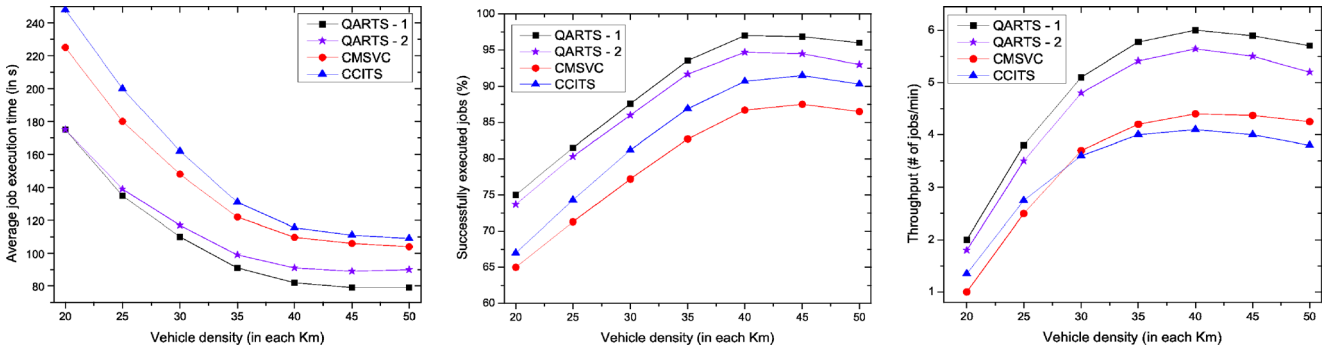
5.3 Simulation results

5.3.1 Impacts of vehicle density

Increasing the number of workers, in general, increases available data processing resources and reduces job execution time. It also increases the interference when transmitting data among the workers and thus increases data transmission time. Figure 4 shows the system performance for varying vehicle densities on the road, ranging from 20-50 vehicles/km. The job arrival rate is fixed at 6 jobs/minute for measuring the system efficiency.

The graphs of Fig. 4a depict that the average execution time of jobs sharply decreases in all the studied approaches with the increasing densities. Such results are achieved through exploiting additional resources in computing tasks on many workers. However, it reaches a saturation point when vehicle density crosses 40. The obtained result clearly reveals the fact that the proposed QARTS-1 and QARTS-2 systems outperform CMSVC and CCITS approaches in average job execution time. This is because QARTS selects vehicles offering reduced execution time and high reliability and thus the average time spent in task execution, task rescheduling, reprocessing, and data re-transmitting can be reduced by a significant amount. However, since QARTS-2 has higher frequency of transmitting heart-beat messages, increasing the message collision in the network with vehicle density. That is why, it causes comparatively low reliability and high job execution time.

In Fig. 4b, the effect of varying vehicle densities on the percentage of jobs executed successfully has been shown. It is depicted that the percentage of successful job execution



(a) Average job execution time (b) Successful job execution (%) (c) System throughput

Fig. 4 Impacts of vehicles' density on performances of the studied task scheduling models

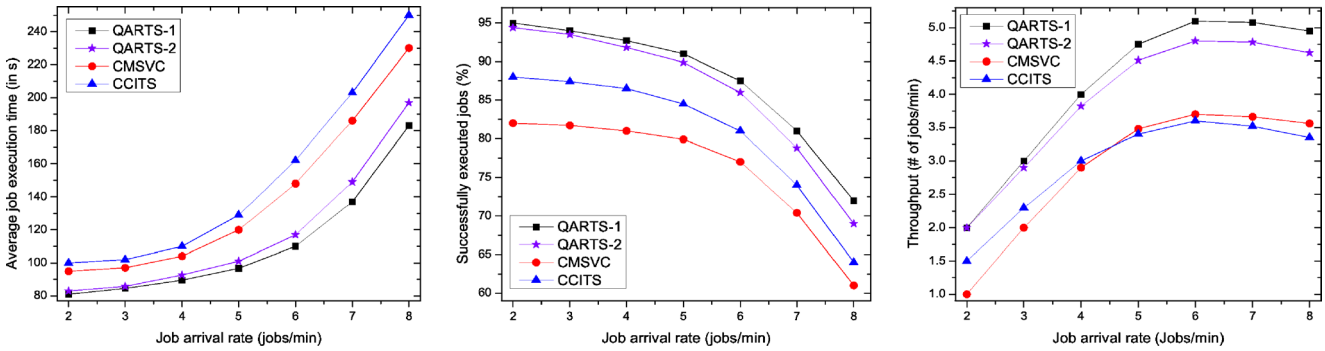
increases exponentially with the vehicle density and starts to decrease after reaching a saturation point. This is caused by the fact that optimal scheduling opportunity increases with the number of vehicles till it reaches approximately 40. However, when the system becomes congested with a large number of vehicles, the number of data transmissions and the delay associated with sending control and data packets increases. The proposed QARTS-1 and QARTS-2 ascertain higher task execution probability, lower task execution and data transmission latencies, and experiences a better percentage in successful job execution. Since heartbeat messages are transmitted at higher rate in QARTS-2, packet collision increases with the vehicle density and thus the successful job execution percentage decreases.

Throughput in terms of number of completed jobs in unit time for varying vehicle densities implementing different task scheduling models is shown in Fig. 4c. The graphs depict the fact that, throughputs of all the task scheduling models increase up to a certain level and then start decreasing slowly for the same reason described above. However, throughput obtained by applying QARTS-1 and QARTS-2 models are higher compared to the existing CMSVC and CCITS systems. The reason behind this result is articulated by the fact that the proposed QARTS system selects workers with minimum cost for data processing and transmission and thus it reduces cost for reprocessing of tasks in a job. The throughput performance of CMSVC system crosses over that of CCITS system with the increasing vehicle densities as the former uses distributed approach opposing to the centralized scheduling policy overhead caused by the later. The QARTS-2 experiences lower throughput achievement for the reasons described above.

5.3.2 Impacts of job arrival rate

As the job arrival rate in the VCC environment increases, the system gradually becomes contented and further performance improvement becomes saturated. Figure 5 shows the system performance for varying job arrival rates at the VCC workers. In this case, we have considered the vehicle density in the road as 30.

The graphs of Fig. 5a states that the task execution delay increases exponentially with the increasing arrival rates in



(a) Average job execution time (b) Successful job execution (%) (c) System throughput

Fig. 5 Impacts of job arrival rates on performances of the studied task scheduling models

all the studied systems. This is because additional computation loads on the vehicular resources cause more time spent in task rescheduling, reprocessing, and data retransmitting compared to time spent in task processing. However, QARTS models have a lower task-execution time compared to CMSVC and CCITS because the former models can reduce the cost of task processing, re-processing, and rescheduling to a great extent by choosing reliable compute resources in the neighborhood. A bit higher task execution delay is observed in QARTS with higher heart-beat message rate due to increased number of re-processing and rescheduling of tasks.

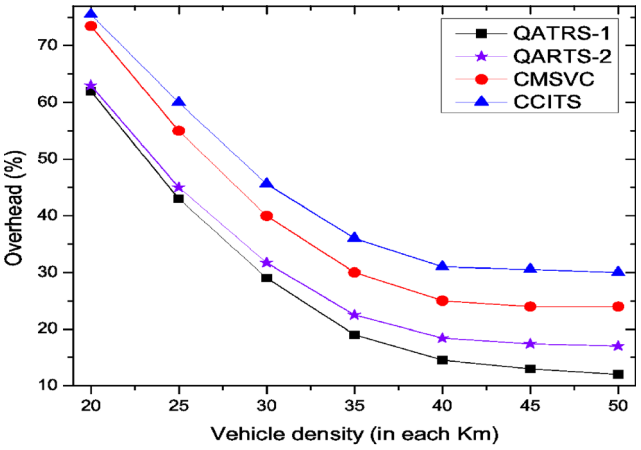
In Fig. 5b, the system performance on the percentage of successfully executed jobs for varying job arrival rates has been shown. The graphs depict the fact that the percentage of job execution decreases exponentially for increasing arrival rates. This is because additional computation load forces the system to choose relatively poor nodes for task execution with respect to reliability and task execution time. However, as the proposed QARTS-1 and QARTS-2 systems select workers considering the success probability of task execution, they experience comparatively better percentage in successful task execution. The QARTS-2 model experiences lower percentage in successful job execution due to the collision of control packets and the situation gets worse with increasing job arrival rates.

The system throughput is shown in Fig. 5c where throughput is derived in terms of number of completed jobs for varying job arrival rates. The figure shows that throughput obtained by applying QARTS-1 and QARTS-2 models are higher compared to those of CMSVC and CCITS systems. The reason behind this is that our QARTS system can serve more tasks in unit time as it can minimize wastage of time in task rescheduling, re-execution, and data re-transmission.

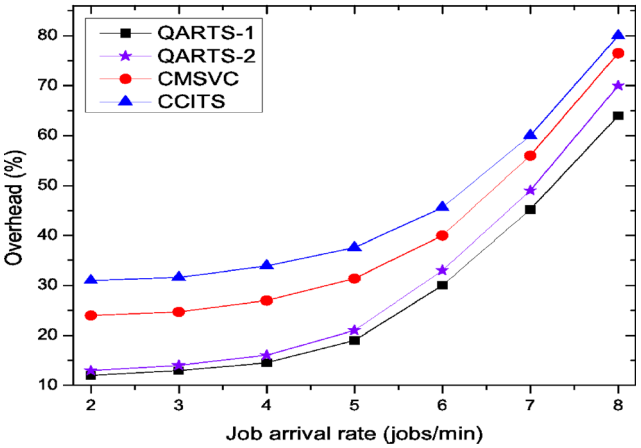
5.3.3 Task execution overhead

Figure 6 demonstrates the task execution overhead of the studied task-scheduling systems against vehicle densities and job arrival rates. For measuring task execution overhead against vehicle densities, job arrival rate is fixed at 6, and the vehicle density is fixed at 30 for varying job arrival rates.

In Fig. 6a, we observe exponential decrease in system overhead incurred when controlling the VCC systems to execute jobs for varying vehicle densities. From the graphs we see that the system is saturated in terms of task execution overhead after reaching the vehicle density of 40. However, QARTS-1 and QARTS-2 have the minimum overhead as they can avoid task rescheduling and re-execution to a large extent and hence they require less time for VCC



(a) Overhead for varying vehicle densities



(b) Overhead for varying job arrival rates

Fig. 6 Task execution overhead of the studied systems

system management. The QARTS-2 has higher overhead for task execution compared to QARTS-1 as the former needs to re-schedule tasks on failure of heart-beat messages.

A comparative performance study among the QARTS, CMSVC, and CCITS systems on the task execution overhead for varying job arrival rates is depicted in Fig. 6b. The graphs show that, in all the strudied systems, the overhead increases exponentially as the job arrival rate increases. However, QARTS-1 and QARTS-2 have the minimum overhead for getting the tasks executed compared to CMSVC and CCITS. The reason behind the behavior is that the QARTS can save time by selecting workers in such a way that the data transmission delay among the workers is minimized and a significant number of tasks do not need to be re-executed. Again, the QARTS-2 experiences a higher overhead compared to QARTS-1 for the reasons described above.

6 Conclusions

Vehicular cloud computing (VCC) provides very significant computational benefits to diverse vehicular applications. However, there remain many obstacles and complexities in its successful implementation. In this article, a reliable task-scheduling model in VCC environment has been developed aiming to minimize execution time so as to satisfy job deadlines. To select an optimal set of vehicular workers, an MILP optimization problem has been initially formulated followed by development of a light weight heuristic task-scheduling algorithm. The simulation performance study proves the effectiveness of our proposed QARTS system in achieving better performances in time and reliability domains compared to the state-of-the-art approaches.

Acknowledgments This work was supported by the Deanship of Scientific Research at King Saud University, Riyadh, Saudi Arabia through the International Research Group Project IRG-14-17.

References

1. Hafeez KA, Zhao L, Ma B, Mark JW (2013) Performance analysis and enhancement of the DSRC for VANET's safety applications. IEEE Trans Veh Technol 62(7):3069–3083
2. Gramaglia M, Calderon M, Bernardos CJ (2014) ABEONA monitored traffic: VANET-assisted cooperative traffic congestion forecasting. IEEE Veh Technol Mag 9(2):50–57
3. Gerla M (2012) Vehicular cloud computing. In: Ad Hoc networking workshop (Med-Hoc-Net), 11th annual mediterranean, pp 152–155
4. Eltoweissy M, Olariu S, Younis M (2010) Towards autonomous vehicular clouds, Ad Hoc networks. Berlin Heidelberg 49:1–16
5. Olariu S, Khalil I, Abuelela M (2011) Taking VANET to the clouds. Int'l J Pervasive Comput Comm 7(1):7–21
6. He W, Yan G, Xu LD (2014) Developing vehicular data cloud services in the IoT environment. IEEE Trans Indus Inf 10(2):1587–1595
7. Lee E, Lee E-K, Gerla M, Oh SY (2014) Vehicular cloud networking: architecture and design principles. IEEE Comm Mag 52(2):148–155
8. Son J, Eun H, Oh H, Kim S, Hussain R (2012) Rethinking vehicular communications: merging VANET with cloud computing. In: IEEE Int'l conf. on cloud computing (CLOUDCOM), pp 606–609
9. Bitam S, Mellouk A (2012) ITS-Cloud: cloud computing for intelligent transportation system. In: Global comm conf. (GLOBECOM). IEEE, pp 2054–2059
10. Aminizadeh L, Yousefi S (2014) Cost minimization scheduling for deadline constrained applications on vehicular cloud infrastructure. In: IEEE Int'l conf. on computer and knowledge engineering (ICCKE), pp 358–363
11. Wang H, Liu RP, Ni W, Chen W, Collings IB (2015) VANET modeling and clustering design under practical traffic, channel and mobility conditions. IEEE Trans Comm 63(3):870–881
12. Das AK, Adhikary T, Razzaque MA, Hong CS (2013) An intelligent approach for virtual machine and QoS provisioning in cloud computing. In: IEEE Int'l conf. on information networking (ICOIN), pp 462–467
13. Adhikary T, Das AK, Razzaque MA, Sarkar AMJ (2013) Energy-efficient scheduling algorithms for data center resources in cloud computing. In: IEEE int'l conf on high performance computing and communications (HPCC), pp 1715–1720. Zhangjiajie
14. Qin Y, Huang D, Zhang X (2012) VehiCloud: cloud computing facilitating routing in vehicular networks, trust. In: IEEE Int'l conf. on security and privacy in computing and communications (TrustCom), pp 1438–1445
15. Abid H, Phuong LTT, Wang J, Lee S, Qaisar S (2011) V-Cloud: vehicular cyber-physical systems and cloud computing. In: Proc of Int'l symposium on applied sciences in biomedical and commun technologies. ACM
16. Yu R, Zhang Y, Gjessing S, Xia W, Yang K (2013) Toward cloud-based vehicular networks with efficient resource management. IEEE Netw 27(5):48–55
17. Basagni S, Conti M, Giordano S, Stojmenovic I (2013) The next paradigm shift: from vehicular networks to vehicular clouds, mobile Ad Hoc networking: the cutting edge directions, 1st edn, Wiley-IEEE Press, pp 645–700
18. Gerla M, Weng J-T, Pau G (2013) Pics-on-wheels: photo surveillance in vehicular cloud. In: Int'l conf on computing, netw. and comm (ICNC), pp 1123–1127
19. Elespuru PR, Shakya S, Mishra S (2009) MapReduce system over heterogeneous mobile devices. In: Int'l workshop on software technologies for embedded and ubiq systems. Springer-Verlag, Heidelberg, pp 168–179
20. Network simulator version 3 (ns-3), https://www.nsnam.org/. Accessed 25 Mar 2015
21. MRPerf simulator, http://research.cs.vt.edu/dssl/mrperf/. Accessed 20 Mar 2015
22. Simulation of urban mobility (SUMO), http://sumo.sourceforge.net. Accessed 18 Mar 2015
23. Lai Y-C, Lin P, Liao W, Chen C-M (2011) A region-based clustering mechanism for channel access in vehicular Ad Hoc networks. IEEE J Selected Areas Comm 29(1):83–93
24. Chai R, Yang B, Li L, Sun X, Chen Q (2013) Clustering-based data transmission algorithms for VANET. In: Int'l conf on wireless comm & sig. proc. (WCSP), pp 1–6
25. Shih HY, Leu JS (2012) Improving resource utilization in a heterogeneous cloud environment. In: Asia-Pacific conference on comm (APCC), pp 185–189
26. NEOS optimization server, http://www.neos-server.org/neos/. Accessed 10 Sep 2015