Authors are encouraged to submit new papers to INFORMS journals by means of a style file template, which includes the journal title. However, use of a template does not certify that the paper has been accepted for publication in the named journal. INFORMS journal templates are for the exclusive purpose of submitting to an INFORMS journal and should not be used to distribute the papers in print or online or to submit the papers to another publication.

Cloud Computing in Space

Jiangchuan Huang
Systems Engineering, UC Berkeley, jiangchuan.huang@gmail.com,

Christoph M. Kirsch
Computer Sciences, University of Salzburg, Austria, ck@cs.uni-salzburg.at,

Raja Sengupta
Systems Engineering, UC Berkeley, sengupta@ece.berkeley.edu,

We apply virtual machine abstractions to networked vehicles enabling what we call cloud computing in space to create performance isolation [23] between customers. In analogy to conventional system virtualization and cloud computing, there are customer-operated virtual vehicles that essentially perform like real vehicles although they are in reality hosted by fewer, shared provider-operated real vehicles. The motion of the virtual vehicles and real vehicles creates migration gain. As a result, cloud computing in space can do better than conventional cloud computing in the sense of realizing high performance isolation (e.g. 98%) while requiring significantly fewer real vehicles (e.g. approximately 1-for-5).

Key words: Cloud computing, virtual vehicle, performance isolation, migration gain

1. Introduction

We explore extending the paradigm of cloud computing to computing tasks having locations in space. A location specific computing task is a triple \((T^a, X, T^S)\) where \(T^a\) is the arrival time of the task, \(X\) its location in space, and \(T^S\) its size. A moving server is considered to serve such a task by traveling to the location \(X\) and staying there for computing time \(T^S\), at any time after the task has arrived, i.e., after time \(T^a\). Sampling applications in time and space, such as those entailed by Google street view, mobile sensor networks [25], or real-time traffic reporting, are examples of this type of computing task. The moving servers, henceforth referred to as real vehicles, are networked vehicles each having all or some of the sensing, computation, communication, and locomotion capabilities. Examples include cars, unmanned aerial vehicles, or helicopters, traveling and then pausing to execute computing tasks like taking pictures, measurements, or monitoring at specified places.
We organize the collection of moving servers as a new type of cloud called the spatial cloud, or the cloud computing in space.

Traditionally, location specific computing tasks were modeled by (i) the vehicle routing problem (VRP) [9], its variations such as (ii) the dynamic travelling repairman problem (DTRP) [3, 12, 13], and (iii) the mobile element scheduling (MES) problem [25]. Tasks are allocated and sequenced based on their arrival times $T_a$, e.g., first come first served [3], locations $X$, e.g., nearest neighbor [3], and sizes $T^S$, e.g., shortest job first [27], to minimize the distance traveled [9], the average time each task spent in the system [3], or the deadline violation ratio [25]. While the current scheduling policies work for the single-customer systems, they do not create performance isolation in multi-customer systems. In the VRP tasks from different customers are indistinguishable to the scheduling policies, meaning a customer submitting more tasks would use more resources, possibly to the detriment of other customers.

To resolve this problem, we formulate the multi-customer VRP by borrowing the concept of a virtual machine from cloud computing. In cloud computing, the virtual machine abstraction creates performance isolation so that resources consumed by one virtual machine do not necessarily harm the performance of other virtual machines [23]. We extend the idea of the virtual machine to define an idea called the virtual vehicle for location specific computing tasks. Just as the cloud computing customer has a service-level agreement (SLA) for a virtual machine, our spatial cloud customer would have an SLA for a virtual vehicle. To a customer, a virtual vehicle is exactly like a real vehicle that travels at the virtual speed specified in the SLA. A customer can buy or reserve a virtual vehicle, and submit location specific tasks to this fixed virtual vehicle, just as she would have done to a real vehicle owned by her. For example, suppose a radio station (the customer) uses a helicopter to overfly accident scenes (the tasks) at 100 mile per hour (mph) for real-time traffic reporting. Such a helicopter would arrive at an accident scene 10 miles away in 6 minutes. We now virtualize this helicopter. Instead of buying or renting, and operating a real helicopter, the radio station would buy a virtual helicopter that travels at 100 mph using an SLA. Our spatial cloud provider would then take responsibility for flying some real helicopter to the accident site in 6 minutes or some near approximation to this time. The service rate of a virtual vehicle is constrained by its virtual speed. A customer does not use more resources by submitting more tasks. This is illustrated in Theorem 1.
The virtual vehicle creates a new kind of service for location specific tasks similar to how the virtual machine creates cloud computing as a new kind of service. Compared to major package delivery companies such as UPS and FedEx, which charge on the urgency and distance of each package (task) delivered and solve the VRP for all the packages, this spatial cloud provider charges on the reservation time of each virtual vehicle. In this way, each customer solves their own VRP and designs a route for their virtual vehicle, just as they would have done for a real vehicle owned by them. Each customer would be able to calculate the expected completion time of their tasks based on the route they designed and the contracted virtual speed. This is given in (4). The spatial cloud provider is freed from designing the VRP route for all the tasks from all the customers. Instead, the provider schedules the real vehicles to serve each task such that they are completed no later than their expected completion time.

A virtual vehicle with a virtual speed works almost as well as a real vehicle with the same speed if a statistically dominant subset [21], e.g., 98%, of the virtual vehicle’s tasks are completed no later than their expected completion time. This ratio must be large enough for the customer to consider herself adequately compensated for the loss by the reduced costs delivered by the spatial cloud. We design the spatial cloud to treat the expected completion time as a “deadline” and call it the virtual deadline. The virtual deadlines make the spatial cloud a soft real-time system [4]. Thus performance metrics such as tardiness and delivery probability defined in (5) and (6) can be used to measure the performance of a virtual vehicle. The spatial cloud achieves high performance isolation if almost every virtual vehicle has a tardiness approaching 0, or a delivery probability approaching 1. We measure the performance isolation by the tardiness and delivery probability both in (8) following [28]. We use Jain’s fairness index [14] to quantify the fairness between virtual vehicles in (9).

We show that the spatial cloud can support a given number of virtual vehicles with significantly fewer real vehicles that travel at the virtual speed while guaranteeing high performance isolation. We quantify the spatial cloud gain by the ratio of the number of virtual vehicles over the number of real vehicles. The gain arises from two phenomena. (i) A customer may not fully utilize her virtual vehicle, enabling the spatial cloud to multiplex several virtual vehicles onto one real vehicle. This type of gain is called multiplexing gain. It is known in communication networks [11] and cloud computing [22]. (ii) The real vehicles
save travel distance by migrating a virtual vehicle hosting a task to another real vehicle closer to the task, creating migration gain. This paper focuses on migration gain since it is unique to this spatial cloud. Migration gain is the reason spatial cloud outperforms conventional cloud. We define the migration cost of a virtual vehicle in (7). Theorem 5 bounds the migration cost.

The software engineering required to build a virtual vehicle was described in [7]. The protocols required to migrate a virtual vehicle was described in [17]. This paper provides the scheduling policies that ensure high performance isolation and spatial cloud gain. We describe the spatial cloud model in Section 2, and analyze it in Section 3 using results in queueing theory [6, 1], stochastic and dynamic vehicle routing [3], and soft real-time systems [4]. Theorem 1 asserts that a customer does not use more resources by submitting more tasks because each virtual vehicle is a $GI/GI/1$ queue [6]. Theorem 2 asserts that each real vehicle is a $\Sigma GI/GI/1$ queue [1] under the Voronoi tessellation [20] in Definition 1, and gives the stability condition. The virtual vehicle and real vehicle queues are depicted in Figure 1. We propose scheduling policies adapted from conventional cloud computing [5] in Section 4. These are the earliest virtual deadline first (EVDF), the earliest dynamic virtual deadline first (EDVDF), both in Definition 5, and the credit scheduling policy in Definition 6. Theorem 3 asserts that the EVDF minimizes tardiness. Theorem 4 identifies a worst-case arrival process maximizing tardiness called the $\eta = 1$ process in Definition 7.

We simulate the system with homogeneous real vehicles (Definition 3) and homogeneous virtual vehicles (Definition 4) under the $\eta = 1$ arrival process under the EVDF, EDVDF and credit scheduling policies in Section 5. This is a worst-case simulation because the $\eta = 1$ arrival process maximizes tardiness. The simulation results are summarized in Figures 4, 5, 6, 7, 8, 9, and Table 2.

2. Model

These are the idealizations for the analysis we do. The spatial cloud is depicted in Figure 1, and is defined as follows: A spatial cloud provider controls $M \in \mathbb{N}$ real vehicles (RVs), $\{RV_m\}_{m=1}^M$, in a convex region $A$ of area $A$. Its customers purchase some $K \in \mathbb{N}$ virtual vehicles (VV$s), $\{VV_k\}_{k=1}^K$. The $K$ VVs must be hosted on the $M$ RVs. Each RV travels at constant speed $v^R$. Each VV has virtual speed $v^V$ in the service-level agreement. To a customer, a virtual vehicle with speed $v^V$ is a replica of a real vehicle with speed $v^R$. It can
generate a sequence of tasks \( \langle \text{Task}_{ki} \rangle_{i=1}^{\infty} \), where \( k \) denotes the \( k \)-th VV and \( i \) the \( i \)-th task generated by it. Each \( \text{Task}_{ki} \) has arrival time \( T_{ki}^{a} \), location \( X_{ki} \), and size \( T_{ki}^{S} \). The sequence \( \langle \text{Task}_{ki} \rangle \) is ordered by the arrival time \( T_{ki}^{a} \). The sequences of tasks generated by different VVs are independent of each other.

Each task arrival process \( \{T_{ki}^{a}\}_{i=1}^{\infty} \) is assumed to be a renewal process. Thus the interarrival time, \( I_{ki}^{a} \equiv T_{ki}^{a} - T_{k(i-1)}^{a} \) is independent and identically distributed (i.i.d.) in \( i \), where \( T_{k0}^{a} \equiv 0 \). The generic interarrival time \( I_{ki}^{a} \) is assumed to be integrable. Thus the task arrival rate of each VV is

\[
\lambda_{k}^{V} = \frac{1}{E[I_{ki}^{a}]} \tag{1}
\]

The task locations \( X_{ki} \) are i.i.d. in \( k \) and \( i \), and uniformly distributed in \( A \) with probability density function (pdf) \( f_{X}(x) = \frac{1}{A}, x \in A \). The task size \( T_{ki}^{S} \) is i.i.d. in \( k \) and \( i \) with pdf \( f_{T^{S}}(t), t \in [0, \infty) \). We denote by \( T^{S} \) the generic term of \( T_{ki}^{S} \). Then \( E[T_{ki}^{S}] = E[T^{S}] \), which is assumed to be finite. Let \( L_{ki} \) denote the distance between \( \text{Task}_{k(i-1)} \) and \( \text{Task}_{ki} \) generated by \( VV_{k} \). Then

\[
L_{ki} = \|X_{ki} - X_{k(i-1)}\| \tag{2}
\]

where \( \| . \| \) is the Euclidean norm defined on region \( A \). \( X_{k0} \) is the initial position of the first RV hosting \( VV_{k} \) given by the provider, which is assumed to be uniformly distributed in \( A \), and independent of \( X_{ki} \). Thus \( L_{ki} \) are i.i.d. in \( k \) and \( i \). We denote by \( L \) the generic term of \( L_{ki} \).

\( VV_{k} \) is assumed to serve the sequence of tasks \( \langle \text{Task}_{ki} \rangle_{i=1}^{\infty} \) under the first come first served (FCFS) policy. It travels to the location \( X_{ki} \) of each task and executes it taking time \( T_{ki}^{S} \). Then the virtual service time of \( \text{Task}_{ki} \), denoted by \( T_{ki}^{V_{serv}} \), includes the travel time and execution time as follows

\[
T_{ki}^{V_{serv}} = \frac{L_{ki}}{v^{V}} + T_{ki}^{S} \tag{3}
\]

Thus \( T_{ki}^{V_{serv}} \) is i.i.d. in \( k \) and \( i \). We denote by \( T^{V_{serv}} \) the generic term of \( T_{ki}^{V_{serv}} \).

Each \( VV_{k} \) is a queue. The virtual completion time \( T_{ki}^{dead} \) of \( \text{Task}_{ki} \) from this queue is

\[
T_{ki}^{dead} = \max \{T_{ki}^{a}, T_{k(i-1)}^{dead} \} + T_{ki}^{V_{serv}} \tag{4}
\]

where \( T_{k0}^{dead} \equiv 0 \). We use the superscript dead for deadline because this virtual completion time is used by our scheduling policies as a task deadline.
Each task, upon arrival, is passed to one of the $M$ real vehicles. Thus we have $M$ real vehicle queues. Each Task$_{ki}$ is served by some real vehicle RV$_m$ as per an allocation policy given in Definition 1 in Section 3.2. The order of service is determined by a scheduling policy. The scheduling policies are discussed in Section 4.

Each Task$_{ki}$ will be completed by an RV at some time $T_{ki}^{comp}$. We assume customer $k$ will be satisfied if $T_{ki}^{comp} \leq T_{ki}^{dead}$. Then the aim of the provider is to achieve $T_{ki}^{comp} \leq T_{ki}^{dead}$ for as many tasks as possible. Thus $T_{ki}^{dead}$ is like a “deadline” for Task$_{ki}$. We call the $T_{ki}^{dead}$ the virtual deadline for Task$_{ki}$. This makes the spatial cloud a soft real-time system [4].

We define the virtual system time of Task$_{ki}$ as $T_{ki}^{Vsys} = T_{ki}^{dead} - T_{ki}^{a}$, and the real system time of Task$_{ki}$ as $T_{ki}^{Rsys} = T_{ki}^{comp} - T_{ki}^{a}$. Thus $T_{ki}^{comp} - T_{ki}^{dead} = T_{ki}^{Rsys} - T_{ki}^{Vsys}$, and $T_{ki}^{comp} \leq T_{ki}^{dead} \Leftrightarrow T_{ki}^{Rsys} \leq T_{ki}^{Vsys}$. When the queue at $VV_k$ is stable, $T_{ki}^{Vsys} \rightarrow T_{ki}^{Vsys}$ in distribution. When the $M$ real vehicle queues are stable, $T_{ki}^{Rsys} \rightarrow T_{ki}^{Rsys}$ in distribution.

We define the relative expected tardiness of $VV_k$ as

$$TD_k = \frac{E\left[\max\left\{T_{ki}^{Rsys} - T_{ki}^{Vsys}, 0\right\}\right]}{E\left[T_{ki}^{Vserv}\right]}$$  (5)

The delivery probability of $VV_k$ is

$$DP_k = P\left(T_{ki}^{Rsys} \leq T_{ki}^{Vsys}\right)$$  (6)

Two consecutive tasks of a VV might be executed by two different RVs. Then the VV migrates from one real vehicle to another. Let $Z_{ki}$ be an indicator set to 1 if $VV_k$ migrates
between Task\(_{k(i-1)}\) and Task\(_{ki}\), and 0 otherwise. When the \(M\) real vehicle queues are stable, \(Z_{ki} \rightarrow Z_k\) in distribution. \(B_{Vk}\) is the number of bits to migrate \(VV_k\) at the migration time. We assume that \(B_{Vk}\) is a constant. \(L\) is the generic distance between two consecutive tasks. We define the inter-virtual deadline time as \(I_{dead}^{ki} = T_{dead}^{ki} - T_{dead}^{k(i-1)}\). Then \(I_{dead}^{ki} \rightarrow I_{dead}^k\) in distribution. The migration cost of \(VV_k\) is

\[
MC_k = B_{Vk} \frac{E[Z_kL]}{E[I_{dead}^k]} \tag{7}
\]

The migration cost has the same unit (bit-meters/second) as in [10].

### 2.1. Performance Isolation

We measure performance isolation by the average of the tardiness and delivery probability following [28].

\[
TD = \frac{1}{K} \sum_{k=1}^{K} TD_k, \quad DP = \frac{1}{K} \sum_{k=1}^{K} DP_k \tag{8}
\]

\(TD = 0\) (resp. \(DP = 1\)) implies \(TD_k = 0\) (resp. \(DP_k = 1\)) for all virtual vehicle \(k\), meaning the system achieves perfect performance isolation. Conversely \(TD \rightarrow \infty\) (resp. \(DP = 0\)) means the system has no performance isolation.

We use Jain’s fairness index [14] to quantify the fairness between virtual vehicles. The fairness indices based on tardiness \(TD_k\) and delivery probability \(DP_k\) are

\[
FI(TD_k) = \frac{\left(\sum_{k=1}^{K} e^{-TD_k}\right)^2}{K \sum_{k=1}^{K} (e^{-TD_k})^2}, \quad FI(DP_k) = \frac{\left(\sum_{k=1}^{K} DP_k\right)^2}{K \sum_{k=1}^{K} DP_k^2} \tag{9}
\]

where we use \(e^{-TD_k}\) to map \(TD_k\) from \([0, \infty)\) to \((0, 1]\). Lower \(TD_k\), or higher \(e^{-TD_k}\), indicates higher performance.

If \(TD_k = TD_l > 0\) (resp. \(DP_k = DP_l > 0\), \(\forall k, l\), then \(FI(TD_k) = 1\) (resp. \(FI(DP_k) = 1\)), indicating completely fair. If \(TD_k > 0\) and \(TD_l = 0\) (resp. \(DP_k > 0\) and \(DP_l = 0\), \(\forall l \neq k\), then \(FI(TD_k) = \frac{1}{K}\) (resp. \(FI(DP_k) = \frac{1}{K}\)), indicating completely unfair. Thus \(FI(TD_k)\) (resp. \(FI(DP_k)\)) ranges between \(\frac{1}{K}\) and 1. The greater the fairness index, the more fair the system. Jain’s fairness index satisfies the desired properties for fairness including population size independence, scale and metric independence, boundedness and continuity. Jain’s fairness index has been used to evaluate virtualization systems [29], [15], [2]. Other performance metrics in this literature include throughput, latency and response time.
2.2. Spatial Cloud Gain

When a provider supports $K$ virtual vehicles with $M$ real vehicles we define the spatial cloud gain $\kappa$ to be

$$\kappa = \frac{K}{M}$$

(10)

The provider gains if $\kappa > 1$. There are two ways a provider can gain:

(i) **Multiplexing gain**: a customer may not fully utilize her virtual vehicle, enabling the provider to multiplex several virtual vehicles onto one real vehicle.

(ii) **Migration gain**: The real vehicles save travel distance by migrating the virtual vehicle generating a task to another real vehicle closer to the task.

The multiplexing gain is observed in communication networks [11] and cloud computing [22]. Migration gain is unique to our cloud computing in space. When every virtual vehicle is fully utilized, there is no multiplexing gain, but there can still be migration gain.

3. Systems

The system has queues at the virtual vehicles and queues at the real vehicles as depicted in Figure 1.

3.1. Virtual Vehicle Queues

The task arrival rate for $VV_k$ is $\lambda^V_k = \frac{1}{E[I^v_k]}$, where $I^v_k$ is the generic interarrival time of $VV_k$.

We define the generic virtual vehicle service rate as

$$\mu^V = \frac{1}{E[T^\text{Vserv}]} = \frac{1}{E[L^v_v] + E[T^S]}$$

(11)

The random process at the output of $VV_k$ is the virtual deadline process $\{T^\text{dead}_{ki}\}_{i=1}^\infty$.

The virtual deadline rate is defined as

$$\lambda^V_{k \text{dead}} = \lim_{i \to \infty} \frac{1}{E[T^\text{dead}_{ki} - T^\text{dead}_{k(i-1)}]}$$

(12)

$VV_k$ is busy if it has a queue and idle if not. Thus $VV_k$ repeats cycles of busy and idle periods. We define $\Theta^V_{kl}$ as the l-th busy period and $I^V_{kl}$ as the l-th idle period. We define the virtual vehicle utilization as

$$u^V_k = \lim_{l \to \infty} \frac{E[\Theta^V_{kl}]}{E[\Theta^V_{kl} + E[I^V_{kl}]]}$$

(13)
A single server queuing system is $GI/GI/1$ if the interarrival times at the input and the service times are positive i.i.d. random variables, separately [6].

The following theorem asserts that when the customer owning a VV creates tasks at a rate less than the service rate, the virtual deadline rate $\lambda_k^V$ is equal to the arrival rate. However, if the customer exceeds the contracted service rate determined by the contracted virtual speed, the virtual deadline rate is equal to the contracted service rate, i.e., the contract throttles the customer’s virtual deadline rate. A higher virtual deadline rate requires more tasks to be completed in unit time. A customer cannot require more than her share of resources by simply generating tasks faster and faster because the virtual deadline rate is throttled by the VV service rate.

**Theorem 1.** Each virtual vehicle $VV_k$ is a $GI/GI/1$ queue. Moreover, 
If $\lambda_k^V < \mu_k^V$, then $u_k^V < 1$ and $\lambda_k^{V\text{dead}} = \lambda_k^V$.
If $\lambda_k^V \geq \mu_k^V$, then $u_k^V = 1$ and $\lambda_k^{V\text{dead}} = \mu_k^V$.

**Proof** (Sketch, see the online supplement for the full proof.) The $GI/GI/1$ follows from the renewal arrival, and the i.i.d. service times $T_{ki}^{V\text{serv}}$. The rest follows from Theorem 1.1 in [6, p. 168]: when $\lambda_k^V < \mu_k^V$ (resp. $\lambda_k^V > \mu_k^V$), the $GI/GI/1$ queue is stable (resp. unstable).

When $\lambda_k^V > \mu_k^V$, the $GI/GI/1$ queue at $VV_k$ is unstable, thus the virtual system time $T_{ki}^{V\text{sys}} = T_{ki}^{\text{dead}} - T_{ki}^{a} \to \infty$ almost surely [6, p. 168]. Since the customer regards a VV as a replica of an RV, we assume the customer will never run the VV in an unstable condition. Thus we assume $\lambda_k^V \leq \mu_k^V$ from now on. Also, when $\lambda_k^V = \mu_k^V$, we assume the customer only generates task arrival processes that result in finite virtual system time, i.e., $T_{ki}^{V\text{sys}} \to T_k^{V\text{sys}}$ in distribution.

### 3.2. Real Vehicle Queues

Each task, upon arrival, is passed to one of the $M$ real vehicles. Each RV has its own subregion, within which the RV runs a scheduling policy to decide which task of which VV to serve when the RV becomes idle. The scheduling policies are discussed in Section 4. We allocate the tasks generated by each VV as follows.

**Definition 1.** The virtual vehicle allocation has three steps:

(i) Divide the region $A$ into $M$ subregions by computing an $M$-median of $A$ that induces a Voronoi tessellation that is equitable with respect to $f_X(x)$ following [20]. An $M$-partition $\{A_m\}_{m=1}^M$ is equitable with respect to $f_X(x)$ if $\int_{A_m} f_X(x) dx = \frac{1}{M}$ for all $m \in \{1, \ldots, M\}$. 


(ii) The real vehicles assign themselves to the subregions in a one-to-one manner.

(iii) Each RV serves the tasks that fall within its own subregion according to some scheduling policy. The VV generating the task is migrated to the RV prior to task execution if the previous task was served by another RV. This migration incurs the cost in (7).

We sequence the tasks contributed by all the virtual vehicles to subregion $A_m$ by their arrival times. The sequence is denoted $\langle Task_{(mj)} \rangle_{j=1}^{\infty}$. Thus $Task_{(mj)}$ is the $j$-th task arrived at real vehicle $m$. Note each $Task_{(mj)}$ corresponds to some $Task_{ki}$ generated by a virtual vehicle $k$ at time $T_{ki}^a$. We write $Task_{(mj)}$ if the task is labeled according to the RV, and $Task_{ki}$ if the task is labeled according to the VV.

Since the task locations $X_{ki}$ are i.i.d. in $k$ and $i$, and uniformly distributed in region $A$, then $X_{(mj)}$ are i.i.d. in $j$ and uniformly distributed in each subregion $A_m$.

We denote by $D_{(m)}$ the distances between two random task locations in subregion $A_m$. Thus,

$$D_{(m)} = \| X_{(mj)} - X_{(ml)} \|$$

where $X_{(mj)}$ and $X_{(ml)}$ are two random task locations in subregion $A_m$.

Let $D_{(mj)}$ denote the distance between $Task_{(mj)}$ and the task executed before it under a scheduling policy $\phi$ in subregion $A_m$. In general, $D_{(mj)}$ is policy dependent. We define a class of policies that produce i.i.d. $D_{(mj)}$ as follows.

**Definition 2.** A scheduling policy $\phi$ in a real vehicle subregion $A_m$ is called *non-location based* if the distance between two consecutively executed tasks is i.i.d.

The common policies in queueing theory such as FCFS, last come first served (LCFS), random order of service (ROS), and shortest job first (SJF) [27] are non-location based policies in the sense of Definition 2. The scheduling policies we propose in Section 4 such as the earliest virtual deadline first (EVDF), the earliest dynamic virtual deadline first (EDVDF) and the credit scheduling are shown to also satisfy Definition 2 by Theorem 3.

Scheduling policies that utilize the location of the tasks such as nearest neighbor (NN) and traveling salesman policy (TSP) on a given set of tasks are location based. The distances between consecutively executed tasks are not independent.

**Definition 3.** The set of $M$ real vehicle subregions $\{A_m\}_{m=1}^{M}$ are said to be *homogeneous* if they all have the same scheduling policy, and $D_{(m)}$ is i.i.d. for all $m = 1, \ldots, M$. 
In the rest of this section and the next, we only consider homogeneous RV subregions under non-location based scheduling policies. We denote by $D$ the generic term of $D_{(m)}$. Then $D_{(mj)}$ is i.i.d. in $m$ and $j$, and has the same distribution as $D$.

The service time of Task$_{(mj)}$ by RV$_m$ is

$$T^{R_{serv}}_{(mj)} = \frac{D_{(mj)}}{v^R} + T^S_{(mj)}$$  \hspace{1cm} (15)

Since $D_{(mj)}$ and $T^S_{(mj)}$ are i.i.d. in $m$ and $j$, separately, then $T^{R_{serv}}_{(mj)}$ is i.i.d. in $m$ and $j$.

We denote by $T^{R_{serv}}$ the generic term of $T^{R_{serv}}_{(mj)}$, and define the generic real vehicle service rate $\mu^R$ as

$$\mu^R = \frac{1}{E[T^{R_{serv}}]} = \frac{1}{E[D]} v^R + E[T^S]$$  \hspace{1cm} (16)

We define

$$\kappa^c = \frac{\mu^R}{\mu^V} = \frac{E[L]}{E[D]} v^R + E[T^S]$$  \hspace{1cm} (17)

Before analyzing the queues at the real vehicles, we list some of the basic results on the thinning and superposition of stochastic processes below for convenience.

(i) Given a renewal process $\{S_n\}$ with rate $\lambda$, let each point $S_n$ for $n = 1, 2, \ldots$ be omitted from the sequence with probability $1 - p$ and retained with probability $p$ for some constant $p$ in $0 < p < 1$, each such point $S_n$ being treated independently. The sequence of retained points, denoted by $\{S^p_n\}$, is called the thinned process with retaining probability $p$. Then $\{S^p_n\}$ is also renewal with rate $p\lambda$ [8, pp. 75-76].

(ii) Let $N_k(t)$ be a stationary process with rate $\lambda_k$, then the superposition of $K$ such independent processes $N(t) = \sum_{k=1}^{K} N_k(t)$ is also stationary with rate $\lambda = \sum_{k=1}^{K} \lambda_k$ [16, Section 14].

(iii) Two stationary stochastic processes are said to be probabilistic replicas of each other if their generic interarrival times are identically distributed [26, p. 21].

The following theorem asserts that the queueing system at each real vehicle is $\Sigma GI/GI/1$ [1] under non-location based policies. This means the arrival process at each real vehicle is the superposition of independent renewal processes and the service time process has positive i.i.d. interarrival times. It also establishes the critical role of $\kappa^c$ in stability of these queues under the assumption that every customer keeps their VV stable, i.e., $\lambda^V_k \leq \mu^V$. 


Theorem 2. Under non-location based scheduling policies, each real vehicle \( RV_m \) is a \( \Sigma GI/GI/1 \) queue with task arrival rate \( \lambda^R = \frac{1}{M} \sum_{k=1}^{K} \lambda^V_k \). Moreover, assume homogeneous real vehicle subregions as in Definition 3. Then

(i) all the real vehicle \( \Sigma GI/GI/1 \) queues are probabilistic replicas of each other, i.e., the interarrival time and service time of each queue are identically distributed, separately.

(ii) Let \( \lambda^V_k \leq \mu^V \). When \( \kappa < \kappa^c \), the \( \Sigma GI/GI/1 \) queue at each real vehicle is stable and \( TD_k \) exists. When \( \kappa > \kappa^c \), the \( \Sigma GI/GI/1 \) queue at each real vehicle is unstable when \( \lambda^V_k = \mu^V \), with \( \kappa \) and \( \kappa^c \) defined in (10) and (17).

Proof (Sketch, see the online supplement for the full proof.) The \( \Sigma GI \) follows from the fact that the arrival process in subregion \( A_m \) is the superposition of \( K \) independent thinned process of \( \{T_{k_1}^a, i=1,\ldots,\infty \} \), which are all renewal. The second \( GI \) follows from the independence of consecutive distances ensured by non-location based policies. (i) follows from the homogeneous RV subregions assumption. (ii) follows from Loynes’ stability condition [18].

4. Scheduling Policies

In this section, we design the scheduling policies inside each RV subregion. We assumed that each task arrival process \( \{T_{k_1}^a\} \) of the \( VV_k \) is renewal with generic interarrival time \( I^a_k \) in Section 2. In this section we further assume that \( I^a_k \) is i.i.d. in \( k \), i.e., the interarrival times \( I^a_{ki} = T^a_{ki} - T^a_{ki(i-1)} \) are i.i.d. in \( k \) and \( i \). We thus denote by \( I^a \) the generic term of \( I^a_k \). Then the task arrival rates are the same for each \( VV \), \( \lambda^V_k = \lambda^V = \frac{1}{E[I^a]} \).

Definition 4. The set of \( K \) virtual vehicles are said to generate homogeneous tasks if the task interarrival times \( I^a_{ki} \), locations \( X_{ki} \) and sizes \( T^S_{ki} \) are all i.i.d. in \( k \) and \( i \), separately, \( k = 1,\ldots, K \) and \( i = 1,2,\ldots \).

In this section we assume that all the VVs generate homogeneous tasks. Then the tardiness \( TD_k \) is the same for all the VVs. Thus by (5) and (8) we have

\[
TD_k = TD = \frac{E\left[\max\{T^{Rsys} - T^{Vsys}, 0\}\right]}{E[T^{Vserv}]}
\]  

(18)

Let \( \Phi \) denote the class of non-location based scheduling policies that are non-preemptive and deadline smooth. A scheduling policy is said to be non-preemptive if under this policy the real vehicle always completes an initiated task even when a priority task enters the
system during service. A scheduling policy is said to be deadline smooth if under this policy the RV serves all the tasks including those whose deadline has passed [19].

Let $TD^\phi$ denote the tardiness as defined by (18) under scheduling policy $\phi \in \Phi$. We consider the following optimization problem:

Find $\psi \in \Phi$ s.t.

$$TD^\psi = \min_{\phi \in \Phi} TD^\phi$$

(19)

We propose the scheduling policies earliest virtual deadline first (EVDF) when the task size is known a priori, its variation earliest dynamic virtual deadline first (EDVDF) when the task size is not known a priori. Both are in Definition 5. Definition 6 is our credit scheduling policy. These scheduling policies are motivated by the simple earliest deadline first and credit scheduler in the cloud computing literature [5]. Our scheduling quantum is the task size, since tasks cannot be preempted. The Xen schedulers [5] have a constant scheduling quantum and can preempt tasks. In cloud computing in space, preemption would waste the time spent traveling to the location.

**Definition 5.** Under the earliest virtual deadline first (EVDF) (resp. earliest dynamic virtual deadline first (EDVDF)) scheduling policy, when a real vehicle becomes idle, the real vehicle always hosts the virtual vehicle whose current task has the earliest virtual deadline as defined in (4) (resp. earliest dynamic virtual deadline as defined in (23)) from the pool of virtual vehicles whose current task is located in the real vehicle subregion, and serves the current task of this chosen virtual vehicle.

**Definition 6.** Under the credit scheduling policy, when a real vehicle becomes idle, the real vehicle always hosts the virtual vehicle with the maximum current credit as described in Section 4.3 from the pool of virtual vehicles whose current task is located in the real vehicle subregion, and serves the current task of this chosen virtual vehicle.

**4.1. Earliest Virtual Deadline First**

The optimality of our EVDF scheduling policy among all the non-location based non-preemptive and deadline smooth scheduling policies follows from the optimality of earliest deadline first [19]. The following theorem shows that our EVDF, EDVDF and credit scheduling policy are in the class of non-location based, non-preemptive, and deadline smooth scheduling policies $\Phi$. Moreover, EVDF minimizes tardiness within this class.
Theorem 3. (i) Let $\phi \in \{EVDF, EDVDF, Credit\}$, as in Definitions 5 and 6, then $\phi \in \Phi$, i.e., $\phi$ is non-location based, non-preemptive, and deadline smooth.

(ii) Assume homogeneous virtual vehicles and homogeneous real vehicle subregions, let $\lambda^R < \mu^R$, with $\lambda^R$ and $\mu^R$ defined in Theorem 2 and (16). Then $TD^{EVDF} = \min_{\phi \in \Phi} TD^\phi$.

Proof (Sketch, see the online supplement for the full proof.)

(i) Non-location based follows from the fact that the EVDF, EDVDF and credit scheduling policies are independent of the locations of the tasks. Non-preemptive and deadline smooth follow from Definitions 5 and 6.

(ii) This follows from the optimality of earliest deadline first [19].

Different task arrival processes will generate different tardiness values. We identify a worst-case arrival process maximizing tardiness. We prove the special case $\eta = 1$ of Definition 7 generates the worst case. This is Theorem 4.

Definition 7. An arrival process $\{T^a_{ki}\}_{i=1}^\infty$ is called an $\eta$-arrival process if

$$T^a_{ki} = \begin{cases} 
0, & i \leq \eta \\
T^{dead}_{k(i-1)}, & i > \eta 
\end{cases}$$

where $\eta \in \mathbb{N}$, and $T^{dead}_{ki}$ is given in (4).

By (4) and (20), $T^a_{ki} \leq T^{dead}_{k(i-1)}$, i.e., each task always arrives no later than the virtual completion time of the previous task. Thus the virtual vehicle is always fully utilized under an $\eta$-arrival process, i.e., the $\eta$-arrival process eliminates multiplexing gain. In particular, when $\eta = 1$, Definition 7 implies the arrival of the current task is the virtual deadline of the previous task. Thus the service times are also the interarrival times and the process at the output of the VV is identical to the arrival process, i.e., it is also an $\eta = 1$ process.

The following theorem establishes the special role of the $\eta = 1$ process. It says all tardiness numbers in Section 5 are worst case.

Theorem 4. Assume homogeneous virtual vehicles and homogeneous real vehicle subregions under the EVDF scheduling policy, let $\kappa \leq \kappa^c$, then the $\eta$-arrival process with $\eta = 1$ for all the virtual vehicles achieves the maximum $TD$ among all the renewal processes.

Proof (Sketch, see the online supplement for the full proof.) This follows from the fact that the $\eta = 1$ process provides the least task information for the EVDF policy among all the renewal processes that fully utilize the virtual vehicles.
We define the travel ratio $r_{tr}$ of VVs as the expected travel time over the expected service time of a task

$$r_{tr} = \frac{E[L]}{E[T_{\text{Vserv}}]} = \frac{E[L]}{v^V} + E[T^S]$$  \hspace{1cm} (21)

The following theorem asserts that the migration cost is bounded.

**Theorem 5.** Under the allocation policy in Definition 1, $MC_k \leq r_{tr} v^V B_V k$, where $B_V k$ is a constant given in (7).

**Proof** (i) By (7), $MC_k = B_V k \frac{E[Z_k L]}{E[T_{\text{dead}}]}$. $Z_k$ indicates migration between two consecutive tasks. Thus $E[Z_k L] \leq E[L]$. By Theorem 1 $\lambda_{V_{\text{dead}}} \leq \mu_V$. Since $\lambda_{V_{\text{dead}}} = \frac{1}{E[T_{\text{dead}}]}$ and $\mu_V = \frac{1}{E[T_{\text{Vserv}}]}$, then $E[Z_k] \geq E[T_{\text{Vserv}}]$. Thus $MC_k \leq B_V k \frac{E[L]}{E[T_{\text{Vserv}}]} = B_V k \frac{E[L]}{E[T_{\text{Vserv}}]} v^V = B_V k r_{tr} v^V$ when substituting (21).

### 4.2. Earliest Dynamic Virtual Deadline First

The task size may not be known a priori in practice as assumed by our EVDF scheduling policy. Therefore we define and evaluate another scheduling policy named earliest dynamic virtual deadline first (EDVDF) which assigns task virtual deadlines based on an estimated task size prior to task completion and updates the task size to the true value after completion. (22) and (23) specify the dynamic virtual deadline computation. $T_{V_{\text{serv}}}^\prime$ denotes the estimated service time of $VV_k$ on Task $k_i$ as

$$T_{V_{\text{serv}}}^\prime = \frac{L_{k_i}}{v^V} + T_{V_{\text{serv}}}^\prime$$  \hspace{1cm} (22)

The dynamic virtual deadline $T_{k_i}^\text{dead}(t)$ is given as follows.

$$T_{k_i}^\text{dead}(t) = \begin{cases} \max \left\{ T^a_{k_i}, T^\text{dead}_{k(i-1)}(t) \right\} + T_{V_{\text{serv}}}^\prime, & t < T_{k_i}^\text{comp} \\ \max \left\{ T^a_{k_i}, T^\text{dead}_{k(i-1)}(t) \right\} + T_{V_{\text{serv}}}, & t \geq T_{k_i}^\text{comp} \end{cases}$$  \hspace{1cm} (23)

where $T_{k_0}^\text{dead} \equiv 0$ and $T_{V_{\text{serv}}}^\prime$ is given in (3).

When Task $k_i$ is completed at $t = T_{k_i}^\text{comp}$ by an RV, $T_{k_i}^S$ is known, $T_{k_i}^\text{dead}(t)$ is updated, the scheduling policy updates all the tasks generated by $VV_k$ following Task $k_i$, i.e., updates $T_{k(i+1)}^\text{dead}(t), T_{k(i+2)}^\text{dead}(t), \ldots$ according to equation (23). The real vehicle then serves the next task with the earliest $T_{k_i}^\text{dead}(t)$. In this way the scheduling policy utilizes the actual task sizes as they become known.

In our implementation of EDVDF in Section 5, the task size estimate is set to be $T_{k_i}^S = E[T_{k_{\text{size}}}^\prime]$, where $\{T_{k_{\text{size}}}^\prime\}$ denotes the sizes of the set of previously executed tasks generated by $VV_k$. $E[T_{k_{\text{size}}}^\prime] = 0$ if $\{T_{k_{\text{size}}}^\prime\}$ is empty. Time-series can also be used to estimate $T_{k_i}^S$.
4.3. Credit Scheduling Policy

We adopt the credit scheduler described in [30] with some changes for our spatial cloud. Under the credit scheduling policy, each VV keeps a balance of credits which can be negative. Each credit has a value of one unit time to be hosted by an RV. A token bucket algorithm [24] is implemented to manage the credits of each VV. Each VV has a bucket. Credits are added to the bucket at constant rate one per unit time, and are expended during service. The bucket can hold at most \( c \) credits. The inflow credits are discarded when the bucket is full.

As shown in Figure 2, VVs are divided into three states: UNDER, with a nonnegative credit balance, OVER, with a negative credit balance, and INACTIVE or halted. The VVs are listed in decreasing order of credit balance.

![Credit scheduling policy diagram](image)

The VV with the most credits is called the current VV, say \( VV_k \). The token bucket algorithm is illustrated in Figure 3. When an RV becomes idle, the RV will travel to and execute the current task generated by \( VV_k \), say \( Task_{ki} \). The scheduler debits \( T_{ki}^{Vserv} \) credits from the bucket of \( VV_k \). \( T_{ki}^{Vserv} \) is calculated according to (22). The scheduler then finds the VV with the maximum credit balance, and the VV with the most credits will become the new current VV. When \( Task_{ki} \) is completed, the scheduler will know the size \( T_{ki}^S \) and compute the true service time \( T_{ki}^{Vserv} \). The consumed credit for \( Task_{ki} \), \( T_{ki}^{Vserv} \), is calculated according to (3). The scheduler returns \( T_{ki}^{Vserv} - T_{ki}^{Vserv} \) credits to \( VV_k \) to adjust the debited credits to be exactly \( T_{ki}^{Vserv} \). This method of debiting \( T_{ki}^{Vserv} \) before execution...
Credit inflow at rate 1
Debit $TV_{serv, ki}$ credits when an RV begins to serve $Task_{ki}$

Maximum credits: $c$

Return $(TV_{serv, ki}' - TV_{serv, ki})$ credits when $Task_{ki}$ is completed

Figure 3 Token bucket algorithm.

and adjusting to $TV_{serv, ki}$ after execution is because the scheduler does not know the size of $Task_{ki}$, and thus does not know $TV_{serv, ki}$, before execution. If the task size $T_{ki}^S$ is known a priori, then $TV_{serv, ki}' = TV_{serv, ki}$. The right amount of credits will be debited at beginning and the adjusted amount equals 0.

5. Experiments

We simulate the system under homogeneous real vehicles and homogeneous virtual vehicles under the EVDF, EDVDF and credit scheduling policies in this section.

5.1. Simulation Setup

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Simulation setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual speed, $v^V$</td>
<td>1 unit distance / unit time</td>
</tr>
<tr>
<td>Real vehicle speed, $v^R$</td>
<td>1 ud/ut</td>
</tr>
<tr>
<td>Size of region $A$</td>
<td>$10 \text{ ud} \times 10 \text{ ud}$</td>
</tr>
<tr>
<td>Task size, $T_{ki}^S$</td>
<td>Uniformly distributed</td>
</tr>
<tr>
<td>Task location, $X_{ki}$</td>
<td>Uniformly distributed in $A$</td>
</tr>
<tr>
<td>Task arrival, ${T_{ki}^a}$</td>
<td>$\eta = 1$ arrival process</td>
</tr>
<tr>
<td>Number of tasks per virtual vehicle</td>
<td>1300</td>
</tr>
<tr>
<td>Tasks used</td>
<td>100th - 600th task of each virtual vehicle</td>
</tr>
<tr>
<td>Scheduling policies</td>
<td>EVDF, EDVDF and Credit</td>
</tr>
</tbody>
</table>

Equations (5), (6), and (17) show the tardiness, delivery probability, and stability threshold $\kappa^c$ are determined by the ratios between the expected task size $E[T^S]$, VV travel time
\[
\frac{E[L]}{v^V}, \text{ and RV travel time } \frac{E[D]}{v^R} \text{ per task, not the absolute values of the three. Thus, in our simulation, we set } v^V = v^R = 1 \text{ unit distance (ud) / unit time (ut). The region } A \text{ is a square of size } 10 \text{ ud} \times 10 \text{ ud. The number of RVs is } M, \text{ with } \sqrt{M} \in \{4, 5, 6, 7, 8, 9, 10\}. \text{ The square region } A \text{ is divided into } M \text{ square subregions, each with edge length } \frac{10}{\sqrt{M}} \text{ ud. The } M \text{ RVs are assigned to the } M \text{ subregions in a one-to-one manner. Each task is uniformly distributed in region } A.\]

Thus \[
\frac{E[L]}{v^V} = \frac{52 \times 10}{1} = 5.2 \text{ ut}, \quad \frac{E[D]}{v^R} = \frac{E[L]}{\sqrt{M}v^V} = \frac{5.2}{\sqrt{M}} \text{ ut}.\]

The task size is also uniformly distributed. We set \([T^S] = r \frac{E[L]}{v^V}\) with \(r \in \{0, 5\%, 10\%, 15\%, 20\%, 25\%\}\), since what matters is the ratio of the time the vehicle is stationary to the time it is traveling, rather than the absolute values. Each RV runs a scheduling policy in its subregion. The scheduling policies include EVDF, EDVDF and credit scheduler. Each virtual vehicle generates tasks with an \(\eta = 1\) arrival process because the \(\eta = 1\) process maximizes tardiness by Theorem 4. The idea is to base any findings on the worst tardiness. The \(\eta = 1\) process also have each virtual vehicle fully utilized by Definition 7. Thus, there is no multiplexing gain. Any spatial cloud gains are derived from migration gain only. We simulate 1300 tasks per VV but calculate the metrics using only the 100-th to the 600-th tasks to ensure the metrics are computed at steady-state. When the 600-th task of each VV is under execution, all the other VVs still have tasks. Thus, every VV is fully utilized from the 100-th to 600-th tasks. Table 1 summarizes the simulation setup.

### 5.2. Simulation Results

Figures 4(a) and 4(b) are the performance isolation metrics, Figures 5(a) and 5(b) are the Jain’s fairness indices, of tardiness and delivery probability under the EVDF policy for different numbers of real and virtual vehicles when the task size \(T^S_{ki} = 0\). Figure 6 is the migration cost, where we set the constant \(B_{V,k}\) in (7) to be one unit. Figure 6 shows that migration cost is bounded as asserted in Theorem 5. To see the spatial cloud gain better, we contour from Figure 4(a) and 4(b) in Figures 4(c) and 4(d) separately. The contours in Figures 4(c) and 4(d) show that the spatial cloud gain \#VV\'s \#RV\'s is significantly high while guaranteeing high performance isolation. For example, 750 VVs are supported with 100 RVs with 1% tardiness in Figure 4(c), while 560 VVs are supported with 100 RVs with 98% delivery probability in Figure 4(d). We also contour from Figure 5(a) and 5(b) in Figures 5(c) and 5(d) separately. The contours in Figures 5(c) and 5(d) show that the fairness indices are very close to one when both the spatial cloud gain and performance isolation are high. For example, with 1% tardiness, 750 VVs are supported with 100 RVs
EVDF: Tardiness – M – κ Relation when η = 1 and T_{0S} = 0

EVDF: Delivery Probability – M – κ Relation when η = 1 and T_{0S} = 0

EVDF: Contour of Tardiness when η = 1 and E[T_{S}] = E[L] / 4V

EVDF: Contour of Delivery Probability when η = 1 and E[T_{S}] = E[L] / 4V

Figure 4  Tardiness and delivery probability with different numbers of RVs and gains under the η = 1 process under the EVDF scheduling policy.
Figure 5  Fairness index with different numbers of RVs and gains under the $\eta = 1$ process under the EVDF scheduling policy.
with fairness index greater than 0.999 in Figure 5(c), while with 98% delivery probability, 560 VVs are supported with 100 RVs with fairness index greater than 0.999 in Figure 5(d). The spatial cloud gain \( \frac{\#VV_s}{\#RV_s} \) can be significantly high while guaranteeing high performance isolation, high fairness, and bounded migration cost.

Second, the spatial cloud has economy of scale (EOS), i.e., the spatial cloud gain \( \frac{\#VV_s}{\#RV_s} \) increases with the number of real vehicles without compromising the performance isolation. This is shown in Figure 4. Figures 4(c) and 4(d) are the case when the task size \( T_{ki}^S = 0 \). Figures 4(e) and 4(f) are the case when the expected task size is 25% of the expected travel time, i.e., \( E[T^S] = E[L]_{4vV}. \) All the curves in Figures 4(c), 4(d), 4(e) and 4(f) are increasing with \#RVs, showing EOS. Also all the curves are upper bounded by the \( \kappa_c \) curve given in (17). This is because the spatial cloud is unstable when \( \kappa = \frac{\#VV_s}{\#RV_s} > \kappa_c \) by Theorem 2. Thus \( \frac{\#VV_s}{\#RV_s} \leq \kappa_c \). The number of VVs admitted should be less than \( \kappa_c \) times the number of RVs. The \( \kappa_c \) value changes with the number of RVs because \( E[D] \) changes with the number of RVs in (17). \( \kappa_c \) is unbounded when \( T_{ki}^S = 0 \). \( \kappa_c < 5 \) when \( E[T^S] = E[L]_{4vV}. \) EOS is also observed in Figures 5(c), 5(d), 5(e), and 5(f) where the performance isolation metrics are replaced with the fairness indices. Figures 5(c) and 5(d) are the case when \( T_{ki}^S = 0 \). Figures 5(e) and 5(f) are the case when \( E[T^S] = E[L]_{4vV}. \)

Third, the spatial cloud gain \( \frac{\#VV_s}{\#RV_s} \) is higher when the task size is smaller. When the task size gets larger, the virtual vehicle spends less time traveling to tasks and more time standing still, and cloud computing in space converges to conventional cloud computing.

We conclude this by comparing the gains in Figures 4(c) and 4(e), and 4(d) and 4(f). See
Table 2. We can see that to guarantee tardiness = 1% with 100 RVs, 750 VVs are supported when task size $T_{ki}^S = 0$, but only 150 VVs are supported when $E[T^S] = \frac{E[L]}{4\nu^v}$. Similarly, to guarantee delivery probability = 95% with 100 RVs, 740 VVs are supported when $T_{ki}^S = 0$, but only 130 VVs are supported when $E[T^S] = \frac{E[L]}{4\nu^v}$. Similar reductions in the number of VVs supported occur when the number of real vehicles is 50 and 80. We have the same conclusion by comparing the gains in Figures 5(c) and 5(e), and 5(d) and 5(f), where the performance isolation metrics are replaced with the fairness indices. Figure 7 plots the reduction in gain as the relative task size $\frac{E[T^S]_{\nu^v}}{E[L]}$ is varied between the two extremes in Table 2. The three curves correspond to three different levels of performance isolation. The spatial cloud can have gain less than one when the task size is large or the number of RVs is small. For example, table 2 shows the $\frac{#VV_s}{#RV_s} < 1$ with 50 RVs if we constrain tardiness to 1% or delivery probability to 95%.

<table>
<thead>
<tr>
<th>#RVs</th>
<th>Tardiness = 1%</th>
<th>Delivery probability = 95%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T_{ki}^S = 0$</td>
<td>$E[T^S] = \frac{E[L]}{4\nu^v}$</td>
</tr>
<tr>
<td>50</td>
<td>225(4.5)</td>
<td>&lt; 50 (&lt; 1)</td>
</tr>
<tr>
<td>80</td>
<td>520(6.5)</td>
<td>112(1.4)</td>
</tr>
<tr>
<td>100</td>
<td>750(7.5)</td>
<td>150(1.5)</td>
</tr>
</tbody>
</table>
Fourth, the EVDF policy has better performance than EDVDF, and EDVDF has better performance than the credit scheduling policy. This supports the optimality of the EVDF policy as asserted in Theorem 3. Figures 8(a) and 8(b) compare the EVDF, EDVDF and credit scheduling policies at the settings used to generate Figures 4(e) and 4(f). We can see that EVDF achieves higher gain than EDVDF, and EDVDF achieves higher gain than credit scheduler for a given number of RVs and a required performance isolation.

Finally, the simulation shows that the spatial cloud is easy to operate. The provider can easily determine the appropriate number of real vehicles for a fixed number of virtual
vehicles and a guaranteed performance isolation because Figures 4(a) and 4(b) show the transition between low and high performance isolation is sharp. Figures 9(a) and 9(b) show the performance isolation by tardiness when different numbers of RVs host 100, 300, and 500 VVs. The scheduling policy is EVDF, and the expected task sizes are $0$ and $E[L_{V}]$. As expected, Figure 9(a) shows the tardiness decreases as the number of RVs increases. More importantly, this change becomes very sharp at 23 RVs on the curve for 100 VVs. The tardiness is very small once $M \geq 28$. The same sharp change occurs when the number of VVs is 300 and 500. The tardiness becomes very small when the number of RVs exceeds 52 and 74, respectively. The same phenomenon appears in Figure 9(b) when the expected task size $E[T^S] = E[L_{V}]$, though the gain diminishes as the task size increases.

6. Conclusion and Future Work

We conclude that (i) a spatial cloud provider can support a given number of virtual vehicles with significantly fewer real vehicles with high performance isolation, high fairness, and bounded migration cost. Moreover, (ii) the spatial cloud has economy of scale, and (iii) the EVDF scheduling policy is in this context optimal. Also, (iv) migration gain happens when the real vehicles spend more time traveling than standing still. As the vehicles spend less time traveling, spatial cloud converges to conventional cloud. Lastly, (v) the spatial cloud is easy to operate.

The mathematical and simulation results in this paper are restricted to customers with renewal task arrivals, and task locations that are i.i.d. and uniform. All virtual vehicles are homogeneous, as are all the real vehicles. Moreover, the real and virtual vehicles have the same speed. If the real vehicles were faster, the results would shift in favor of spatial cloud computing and vice versa. Non-uniformly distributed task locations could reduce migration gain. For example, if all tasks are generated at a single point, migration gain would vanish. Any gain would arise only from multiplexing and the spatial cloud would be as the conventional cloud. The focus on uniformly distributed locations brings out the differences between the two clouds. The non-homogeneous real or virtual vehicle case opens a larger parameter space. Future work can be done to investigate the non-uniform task distribution and non-homogeneous real or virtual vehicle cases.

Acknowledgments

Research supported in part by the National Science Foundation (CNS1136141) and by the National Research Network RiSE on Rigorous Systems Engineering (Austrian Science Fund S11404-N23).
References


Cloud Computing in Space

Jiangchuan Huang
Systems Engineering, UC Berkeley, jiangchuan.huang@gmail.com,

Christoph M. Kirsch
Computer Sciences, University of Salzburg, Austria, ck@cs.uni-salzburg.at,

Raja Sengupta
Systems Engineering, UC Berkeley, sengupta@ee.berkeley.edu,

We apply virtual machine abstractions to networked vehicles enabling what we call cloud computing in space to create performance isolation [10] between customers. In analogy to conventional system virtualization and cloud computing, there are customer-operated virtual vehicles that essentially perform like real vehicles although they are in reality hosted by fewer, shared provider-operated real vehicles. The motion of the virtual vehicles and real vehicles creates migration gain. As a result, cloud computing in space can do better than conventional cloud computing in the sense of realizing high performance isolation (e.g. 98%) while requiring significantly fewer real vehicles (e.g. approximately 1-for-5).

Key words: Cloud computing, virtual vehicle, performance isolation, migration gain

Acknowledgments
Research supported in part by the National Science Foundation (CNS1136141) and by the National Research Network RiSE on Rigorous Systems Engineering (Austrian Science Fund S11404-N23).

Appendix A: Proofs of the Theorems
For the convenience of the reader, we restate some of the definitions in the article. When a spatial cloud provider supports $K$ virtual vehicles (VVs) with $M$ real vehicles (RVs) in a convex region $A$, we define the spatial cloud gain $\kappa$ to be

$$\kappa = \frac{K}{M} \quad (1)$$

We define

$$\kappa^c = \frac{\mu^r}{\mu^v} = \frac{\frac{\kappa |L|}{\rho} + E[T^S]}{\frac{\kappa |D|}{\rho} + E[T^S]} \quad (2)$$

where $\mu^v$ is the generic virtual vehicle service rate defined in (6), and $\mu^r$ is the generic real vehicle service rate defined in (12).
Virtual completion time is used by our scheduling policies as a task deadline. Where
\( T \) given in Definition 1. The order of service is determined by a scheduling policy.

Thus, if \( T \) spatial cloud a soft real-time system \([2]\).

Each \( VV_k \) is a queue. The virtual completion time \( T_{ki}^{dead} \) of \( Task_{ki} \) from this queue is

\[
T_{ki}^{dead} = \max \{ T_{ki}, T_{ki}^{dead}(i-1) \} + T_{ki}^{Vserv}
\]

where \( T_{ki}^{dead} \equiv 0 \), and \( T_{ki} \) is the arrival time of \( Task_{ki} \). We use the superscript dead for deadline because this virtual completion time is used by our scheduling policies as a task deadline.

When the task size is not known a priori, we define the dynamic virtual deadline \( T_{ki}^{dead}(t) \) as follows.

\[
T_{ki}^{dead}(t) = \begin{cases} 
\max \{ T_{ki}, T_{ki}^{dead}(t-1) \} + T_{ki}^{Vserv}, & t < T_{ki}^{comp} \\
\max \{ T_{ki}, T_{ki}^{dead}(t-1) \} + T_{ki}^{Vserv}, & t \geq T_{ki}^{comp}
\end{cases}
\]

where \( T_{ki}^{dead} \equiv 0 \) and \( T_{ki}^{Vserv} \) is given in (3). \( T_{ki}^{comp} \) is the completion time of \( Task_{ki} \). \( T_{ki}^{Vserv} \) is the estimated service time of \( VV_k \) on \( Task_{ki} \). \( T_{ki}^{Vserv} \) denotes the task size estimate. For example, we can set \( T_{ki}^{Vserv} = E[T_{ki}^{Serv}] \), where \( \{ T_{ki}^{Serv} \} \) denotes the sizes of the set of previously executed tasks generated by \( VV_k \).

The task arrival rate for \( VV_k \) is \( \lambda_k^V = \frac{1}{E[T_k]} \), where \( T_k \) is the generic interarrival time of \( VV_k \).

We define the generic virtual vehicle service rate as

\[
\mu_k^V = \frac{1}{E[T_{Vserv}]} = \frac{1}{E[T_k^V] + E[T_k^S]}
\]

Each \( Task_{ki} \) will be completed by an RV at some time \( T_{ki}^{comp} \). We assume customer \( k \) will be satisfied if \( T_{ki}^{comp} \leq T_{ki}^{dead} \). Then the aim of the provider is to achieve \( T_{ki}^{comp} \leq T_{ki}^{dead} \) for as many tasks as possible. Thus \( T_{ki}^{dead} \) is like a “deadline” for \( Task_{ki} \). We call the \( T_{ki}^{dead} \) the virtual deadline for \( Task_{ki} \). This makes the spatial cloud a soft real-time system \([2]\).

We define the virtual system time of \( Task_{ki} \) as \( T_{ki}^{Vsys} = T_{ki}^{dead} - T_{ki}^{a} \), and the real system time of \( Task_{ki} \) as \( T_{ki}^{Rsys} = T_{ki}^{comp} - T_{ki}^{a} \). Thus \( T_{ki}^{comp} - T_{ki}^{dead} = T_{ki}^{Rsys} - T_{ki}^{Vsys} \), and \( T_{ki}^{comp} \leq T_{ki}^{dead} \iff T_{ki}^{Rsys} \leq T_{ki}^{Vsys} \). When the queue at \( VV_k \) is stable, \( T_{ki}^{Vsys} \rightarrow T_{ki}^{Vsys} \) in distribution. When the \( M \) real vehicle queues are stable, \( T_{ki}^{Rsys} \rightarrow T_{ki}^{Rsys} \) in distribution.

We define the relative expected tardiness of \( VV_k \) as

\[
TD_k = \frac{E[\max\{T_{ki}^{Rsys} - T_{ki}^{Vsys}, 0\}]}{E[T_{Vserv}]}
\]

The delivery probability of \( VV_k \) is

\[
DP_k = P(T_{ki}^{Rsys} \leq T_{ki}^{Vsys})
\]

Each \( Task_{ki} \), upon arrival, is served by some real vehicle \( RV_m \), \( m = 1, \ldots, M \), as per an allocation policy given in Definition 1. The order of service is determined by a scheduling policy.
 Definition 1. The virtual vehicle allocation has three steps:

(i) Divide the region \( A \) into \( M \) subregions by computing an \( M \)-median of \( A \) that induces a Voronoi tessellation that is equitable with respect to \( f_X(x) \) following [9]. An \( M \)-partition \( \{ A_m \}_{m=1}^M \) is equitable with respect to \( f_X(x) \) if \( \int_{A_m} f_X(x) \, dx = \frac{1}{M} \) for all \( m \in \{1, \ldots, M\} \).

(ii) The real vehicles assign themselves to the subregions in a one-to-one manner.

(iii) Each RV serves the tasks that fall within its own subregion according to some scheduling policy. The RV generating the task is migrated to the RV prior to task execution if the previous task was served by another RV. This migration incurs the cost in (9).

The migration cost of \( VV_k \) is

\[
MC_k = B_{V_k} \frac{E[Z_k L]}{E[I_k^{dead}]} \tag{9}
\]

where \( B_{V_k} \) is the number of bits to migrate \( VV_k \) at the migration time. \( Z_k \) indicates whether the migration happens or not. \( I_k^{dead} = T_k^{dead} - T_{k(i-1)}^{dead} \) is the inter-virtual deadline time. And \( I_k^{dead} \to I_k^{dead} \) in distribution.

The migration cost has the same unit (bit-meters/second) as in [5].

We sequence the tasks contributed by all the virtual vehicles to subregion \( A_m \) by their arrival times. The sequence is denoted \( \langle Task_{(m_j)} \rangle_{j=1}^\infty \). Thus \( Task_{(m_j)} \) is the \( j \)-th task arrived at real vehicle \( m \). Note each \( Task_{(m_j)} \) corresponds to some \( Task_{k_i} \) generated by a virtual vehicle \( k \) at time \( T_{k_i} \). We write \( Task_{(m_j)} \) if the task is labeled according to the RV, and \( Task_{k_i} \) if the task is labeled according to the VV.

Since the task locations \( X_{k_i} \) are i.i.d. in \( k \) and \( i \), and uniformly distributed in region \( A \), then \( X_{(m_j)} \) are i.i.d. in \( j \) and uniformly distributed in each subregion \( A_m \).

We denote by \( D_{(m)} \) the distances between two random task locations in subregion \( A_m \). Thus,

\[
D_{(m)} = \| X_{(m_j)} - X_{(m_l)} \| \tag{10}
\]

where \( X_{(m_j)} \) and \( X_{(m_l)} \) are two random task locations in subregion \( A_m \).

Let \( D_{(m_j)} \) denote the distance between \( Task_{(m_j)} \) and the task executed before it under a scheduling policy \( \phi \) in subregion \( A_m \). In general, \( D_{(m_j)} \) is policy dependent. We define a class of policies that produce i.i.d. \( D_{(m_j)} \) as follows.

Definition 2. A scheduling policy \( \phi \) in a real vehicle subregion \( A_m \) is called non-location based if the distance between two consecutively executed tasks is i.i.d.

Definition 3. The set of \( M \) real vehicle subregions \( \{ A_m \}_{m=1}^M \) are said to be homogeneous if they all have the same scheduling policy, and \( D_{(m)} \) is i.i.d. for all \( m = 1, \ldots, M \).

The service time of \( Task_{(m_j)} \) by \( RV_m \) is

\[
T_{(m_j)}^{Serv} = \frac{D_{(m_j)}}{\mu_R} + T_{(m_j)}^S \tag{11}
\]

Since \( D_{(m_j)} \) and \( T_{(m_j)}^S \) are i.i.d. in \( m \) and \( j \), separately, then \( T_{(m_j)}^{Serv} \) is i.i.d. in \( m \) and \( j \). We denote by \( T_{(m_j)}^{Serv} \) the generic term of \( T_{(m_j)}^{Serv} \), and define the generic real vehicle service rate \( \mu_R \) as

\[
\mu_R = \frac{1}{E[T_{(m_j)}^{Serv}]} = \frac{1}{E[D]_{\phi} + E[T^S]} \tag{12}
\]
Definition 4. The set of $K$ virtual vehicles are said to generate homogeneous tasks if the task interarrival times $I_{ki}$, locations $X_{ki}$ and sizes $T_{ki}$ are all i.i.d. in $k$ and $i$, separately, $k = 1, \ldots, K$ and $i = 1, 2, \ldots$.

We propose the scheduling policies earliest virtual deadline first (EVDF) when the task size is known a priori, its variation earliest dynamic virtual deadline first (EDVDF) when the task size is not known a priori. Both are in Definition 5. Definition 6 is our credit scheduling policy.

Definition 5. Under the earliest virtual deadline first (EVDF) (resp. earliest dynamic virtual deadline first (EDVDF)) scheduling policy, when a real vehicle becomes idle, the real vehicle always hosts the virtual vehicle whose current task has the earliest virtual deadline as defined in (4) (resp. earliest dynamic virtual deadline as defined in (5)) from the pool of virtual vehicles whose current task is located in the real vehicle subregion, and serves the current task of this chosen virtual vehicle.

Definition 6. Under the credit scheduling policy, when a real vehicle becomes idle, the real vehicle always hosts the virtual vehicle with the maximum current credit as described in Section 4.3 in the article from the pool of virtual vehicles whose current task is located in the real vehicle subregion, and serves the current task of this chosen virtual vehicle.

A.1. Proof of Theorem 1

A single server queuing system is $GI/GI/1$ if the interarrival times at the input and the service times are positive i.i.d. random variables, separately [3].

Theorem 1 asserts that when the customer owning a VV creates tasks at a rate less than the service rate, the virtual deadline rate $\lambda_k^V$ is equal to the arrival rate. However, if the customer exceeds the contracted service rate determined by the contracted virtual speed, the virtual deadline rate is equal to the contracted service rate, i.e., the contract throttles the customer’s virtual deadline rate. A higher virtual deadline rate requires more tasks to be completed in unit time. A customer cannot require more than her share of resources by simply generating tasks faster and faster because the virtual deadline rate is throttled by the VV service rate.

Theorem 1 Each virtual vehicle $VV_k$ is a $GI/GI/1$ queue. Moreover,

If $\lambda_k^V < \mu^V$, then $u_k^V < 1$ and $\lambda_k^{V\text{dead}} = \lambda_k^V$.

If $\lambda_k^V \geq \mu^V$, then $u_k^V = 1$ and $\lambda_k^{V\text{dead}} = \mu^V$.

Proof By assumption the task arrival process of $VV_k$ is renewal. Thus the interarrival times are positive and i.i.d.. Also the service times $T_{ki}^{V\text{serv}}$ are positive and i.i.d.. Thus each virtual vehicle $VV_k$ is a $GI/GI/1$ queue.

(i) When $\lambda_k^V < \mu^V$, the $GI/GI/1$ queue is stable by Theorem 1.1 in [3, p. 168]. The number of tasks waiting in the queue is finite almost surely, the expected interdeparture time $E[T_{ki}^{V\text{dead}} - T_{(k-1)i}^{V\text{dead}}]$ of the VV is $\frac{1}{\mu^V}$ because no tasks are lost and no extra task is created. Thus $\lambda_k^{V\text{dead}} = \lambda_k^V$. Stability also ensures that $\lim_{m \to \infty} E[\Theta_k^V] = E[\Theta_k^V]$ and $\lim_{m \to \infty} E[I_k^V] = E[I_k^V]$, where $\Theta_k^V$ and $I_k^V$ are the generic busy period and idle period. Thus $u_k^V = \frac{E[\Theta_k^V]}{E[\Theta_k^V] + E[I_k^V]}$. According to [11, p. 21], $u_k^V = \frac{\lambda_k^V}{\mu^V} \times \frac{1}{\mu^V}$. Since no tasks are lost or created in the system, $u_k^V = \frac{\lambda_k^V}{\mu^V} < 1$. 


(ii) When $\lambda_k^V > \mu^V$, the $GI/GI/1$ queue is unstable by Theorem 1.1 in [3, p. 168], the number of tasks waiting in the queue goes to infinity as time goes to infinity. The busy period tends to infinity and the idle period tends to 0. Thus $u_k^V = \lim_{t \to \infty} \frac{E[\Theta_k^V]}{E[\Theta_k^V] + E[\tau_k^V]} = 1$, and the interdeparture time equals the VV service time. Then $\lambda_k^{V\text{dead}} = \mu^V$.

(iii) When $\lambda_k^V = \mu^V$, the number of tasks waiting in the queue can either be finite, or goes to infinity as time goes to infinity depending on the arrival process $\{T_k^V\}$. So this case either goes to case (i) or (ii). In either case, we have $u_k^V = 1$ and $\lambda_k^{V\text{dead}} = \mu^V$.

When $\lambda_k^V > \mu^V$, the $GI/GI/1$ queue at $VV_k$ is unstable, thus the virtual system time $T_k^{V\text{sys}} = T_k^{\text{dead}} - T_k^V \to \infty$ almost surely [3, p. 168]. Since the customer regards a VV as a replica of an RV, we assume the customer will never run the VV in an unstable condition. Thus we assume $\lambda_k^V \leq \mu^V$ from now on. Also, when $\lambda_k^V = \mu^V$, we assume the customer only generates task arrival processes that result in finite virtual system time, i.e., $T_k^{V\text{sys}} \to T_k^{V\text{sys}}$ in distribution.

A.2. Proof of Theorem 2

Before proving Theorem 2, we list some of the basic results on the thinning and superposition of stochastic processes below for convenience.

(i) Given a renewal process $\{S_n\}$ with rate $\lambda$, let each point $S_n$ for $n = 1, 2, \ldots$ be omitted from the sequence with probability $1 - p$ and retained with probability $p$ for some constant $p$ in $0 < p < 1$, each such point $S_n$ being treated independently. The sequence of retained points, denoted by $\{S'_n\}$, is called the thinned process with retaining probability $p$. Then $\{S'_n\}$ is also renewal with rate $p\lambda$ [4, pp. 75-76].

(ii) Let $N_k(t)$ be a stationary process with rate $\lambda_k$, then the superposition of $K$ such independent processes $N(t) = \sum_{k=1}^{K} N_k(t)$ is also stationary with rate $\lambda = \sum_{k=1}^{K} \lambda_k$ [6, Section 14].

(iii) Two stationary stochastic processes are said to be probabilistic replicas of each other if their generic interarrival times are identically distributed [11, p. 21].

Theorem 2 asserts that the queueing system at each real vehicle is $\Sigma GI/GI/1$ [1] under non-location based policies. This means the arrival process at each real vehicle is the superposition of independent renewal processes and the service time process has positive i.i.d. interarrival times. It also establishes the critical role of $\kappa^c$ given in (2) in stability of these queues under the assumption that every customer keeps their VV stable, i.e., $\lambda_k^V \leq \mu^V$.

**Theorem 2** Under non-location based scheduling policies, each real vehicle $RV_m$ is a $\Sigma GI/GI/1$ queue with task arrival rate $\lambda^R = \frac{1}{M} \sum_{k=1}^{K} \lambda_k^V$. Moreover, assume homogeneous real vehicle subregions as in Definition 3. Then

(i) all the real vehicle $\Sigma GI/GI/1$ queues are probabilistic replicas of each other, i.e., the interarrival time and service time of each queue are identically distributed, separately.

(ii) Let $\lambda_k^V \leq \mu^V$. When $\kappa < \kappa^c$, the $\Sigma GI/GI/1$ queue at each real vehicle is stable and $TD_k$ exists. When $\kappa > \kappa^c$, the $\Sigma GI/GI/1$ queue at each real vehicle is unstable when $\lambda_k^V = \mu^V$, with $\kappa$ and $\kappa^c$ defined in (1) and (2).
Proof  
(i) By Definition 1, our $M$-Voronoi tessellation creates equitable subregions and $A_m$ is a Voronoi subregion. Then each task in the sequence $\langle Task_{ki}\rangle_{i=1}^{\infty}$ falls in subregion $A_m$ with probability $\frac{1}{M}$. Hence the arrival time of tasks in the sequence $\langle Task_{ki}\rangle_{i=1}^{\infty}$ that fall in $A_m$ is a thinned process of $\{T^a_{ki}\}_{i=1}^{\infty}$ with retaining probability $p = \frac{1}{M}$. We denote the thinned arrival process as $\{T^{ap}_{ki}\}_{i=1}^{\infty}$. Since $\{T^{ap}_{ki}\}_{i=1}^{\infty}$ is renewal with rate $\lambda_k^V$, then $\{T^{ap}_{ki}\}_{i=1}^{\infty}$ is renewal with rate $\frac{\lambda_k^V}{M}$ by Example 4.3(a) in [4, pp. 75-76]. Thus the arrival process in subregion $A_m$, $\{T^a_{(mj)}\}_{j=1}^{\infty}$, is the superposition of $\{T^{ap}_{ki}\}_{i=1}^{\infty}$, $k = 1, \ldots, K$, or $K$ independent renewal processes. This proves the second $GI$. A renewal process is also stationary, so $\{T^a_{(mj)}\}_{j=1}^{\infty}$ is stationary, and the arrival rate at $RV_m$ is $\lambda^R = \frac{\sum_{k=1}^{K} \lambda_k^V}{M}$ by [6, Section 14].

Moreover, the $M$ thinned processes $\{T^{ap}_{ki}\}_{i=1}^{\infty}$ generated from the same $VV_k$ arrival process are probabilistic replicas of each other because the retaining probabilities are all $\frac{1}{M}$. Since the arrival process of each RV subregion, $\{T^a_{(mj)}\}_{j=1}^{\infty}$, is the superposition of thinned replicas from each $VV$, then $\{T^a_{(mj)}\}_{j=1}^{\infty}$ are i.i.d. in $m$. Homogeneous RV subregions ensures that the service times at $RV_m$, $T^{Rserv}_{(mj)}$, are i.i.d. in $m$ and $j$. Thus all the real vehicle $GI/GI/1$ queues are probabilistic replicas of each other. (i) follows.

(ii) When $\lambda_k^V \leq \mu^V$, when $\kappa < \kappa^c$, $\lambda^R = \frac{\sum_{k=1}^{K} \lambda_k^V}{M} \leq \frac{\kappa \mu^V}{M} = \kappa \mu^V < \kappa^c \mu^V = \mu^R$, thus each $GI/GI/1$ queue at $RV_m$ is stable by Loynes’ stability condition [7]. Task$_{ki}$ will be served by one of the RVs, say Task$_{ki}$, is the $j$-th task served by $RV_m$. Then Task$_{ki}$ corresponds to Task$_{(mj)}$, we denote by Task$_{ki(m)}$ for the same task. Since $T^a_{k(i-1)} \leq T^a_k$ and $T^a_{(m(j-1))} \leq T^a_{(mj)}$, then $i \to \infty \Leftrightarrow T^a_k \to \infty \Leftrightarrow T^a_{(mj)} \to \infty \Leftrightarrow j \to \infty$. Thus

$$T^{Rsys}_{k(m)} \to p T^{Rsys}_{k(m)} \tag{13}$$

Since the all the RV $GI/GI/1$ queues are probabilistic replicas of each other, then $T^{Rsys}_{k(m)}$ is i.i.d. in $m$, we denote by $T^{Rsys}_k$ the generic term of $T^{Rsys}_{k(m)}$. Since a task from $VV_k$ can fall in any of the RV subregions with probability $\frac{1}{M}$, thus (13) becomes

$$T^{Rsys}_{k(m)} \to p T^{Rsys}_k \tag{14}$$

Also, when $\lambda_k^V \leq \mu^V$, the $GI/GI/1$ queue at each $VV$ is stable, $T^{Vsys}_{k(m)} \to T^{Vsys}_{k(m)}$ in distribution. When $\lambda_k^V = \mu^V$, by our assumption that follows Theorem 1, the customer will provide arrival process that guarantees $T^{Vsys}_{k(m)} \to T^{Vsys}_{k(m)}$ in distribution. Thus both $T^{Rsys}_k$ and $T^{Vsys}_k$ converges to $T^{Rsys}_k$ and $T^{Vsys}_k$ in distribution. The tardiness $TD_k$ defined in (15) exists.

We define the relative expected tardiness of $VV_k$ as

$$TD_k = E \left[ \max \left\{ T^{Rsys}_k - T^{Vsys}_k, 0 \right\} \right] \tag{15}$$

We measure performance isolation by the average of the tardiness and delivery probability following [12].

$$TD = \frac{1}{K} \sum_{k=1}^{K} TD_k, \quad DP = \frac{1}{K} \sum_{k=1}^{K} DP_k \tag{16}$$

$TD = 0$ (resp. $DP = 1$) implies $TD_k = 0$ (resp. $DP_k = 1$) for all virtual vehicle $k$, meaning the system achieves perfect performance isolation. Conversely $TD \to \infty$ (resp. $DP = 0$) means the system has no performance isolation.
We assume that all the VVs generate homogeneous tasks. Then the tardiness $TD_k$ is the same for all the VVs. Thus by (15) and (16) we have

$$TD_k = TD = \frac{E[\max\{T^{Rsys} - T^{Vsys}, 0\}]}{E[T^{Vserv}]}$$

(17)

When $\kappa > \kappa^c$ and $\lambda^V_k = \mu^V$, $\lambda^R = \sum_{k=1}^{K} \lambda^V_k = \sum_{k=1}^{K} \mu^V = \kappa \mu^V > \kappa^c \mu^V = \mu^R$. Thus each $\Sigma GI/GI/1$ queue at $RV_m$ is unstable by Loynes’ stability condition [7].

A.3. Proof of Theorem 3

Let $\Phi$ denote the class of non-location based scheduling policies that are non-preemptive and deadline smooth. A scheduling policy is said to be non-preemptive if under this policy the real vehicle always completes an initiated task even when a priority task enters the system during service. A scheduling policy is said to be deadline smooth if under this policy the RV serves all the tasks including those whose deadline has passed [8].

The optimality of our EVDF scheduling policy among all the non-location based non-preemptive and deadline smooth scheduling policies follows from Theorem 1 of [8]. We restate this result for convenience below.

Theorem 1 of [8]: For any convex function $g: \mathbb{R} \to \mathbb{R}$, $E[g(R^\phi)] \leq E[g(R^\psi)]$ whenever $\phi \ll \psi$.

Theorem 1 of [8] assumes a $G/GI/1$ queue served by non-preemptive and deadline smooth scheduling policies. The $G$ in a $G/GI/1$ queue means the task arrival process is stationary and ergodic. $\phi$ and $\psi$ denote two admissible non-preemptive and deadline smooth scheduling policies. $\phi \ll \psi$ when $\phi$ always chooses a customer having a deadline earlier than that of the customer chosen by $\psi$. In particular the earliest deadline first (EDF) scheduling policy always gives priority to the customer having the earliest deadline, and the latest deadline first (LDF) one gives priority to the customer having the latest deadline. Then, by definition $EDF \ll \phi \ll LDF$ for any admissible scheduling policy $\phi$. $R^\phi$ is the steady-state value of $R_n = D_n - T_n - W_n$ under policy $\phi$, where $D_n$, $T_n$ and $W_n$ are the deadline, arrival time and waiting time of the $n$-th task.

Theorem 3 shows that our EVDF, EDVDF and credit scheduling policy are in the class of non-location based, non-preemptive, and deadline smooth scheduling policies $\Phi$. Moreover, EVDF minimizes tardiness within this class.

**Theorem 3** (i) Let $\phi \in \{EVDF, EDVDF, Credit\}$, as in Definitions 5 and 6, then $\phi \in \Phi$, i.e., $\phi$ is non-location based, non-preemptive, and deadline smooth.

(ii) Assume homogeneous virtual vehicles and homogeneous real vehicle subregions, let $\lambda^R < \mu^R$, with $\lambda^R$ and $\mu^R$ defined in Theorem 2 and (12). Then $TD^{EVDF} = \min_{\phi \in \Phi} TD^\phi$.

**Proof** (i) The EVDF, EDVDF and credit scheduling policies schedule only based on virtual deadlines, dynamic virtual subregion, each task and credit of each VV, separately. They are independent of the distance between two consecutively executed tasks, $D_{(m,j)}$. Thus $D_{(m,j)}$ is the distance between two random task locations in subregion $A_\alpha$. Thus $D_{(m,j)}$ is i.i.d. in $j$ and has the same distribution as $D_{(m)}$. Thus the three policies are non-location based. The three policies always serve all tasks, even if deadlines have passed. Thus they are smooth with respect to the virtual deadlines as defined in (4). The three policies always completes an initiated task even when a priority task enters the system in the meanwhile. Thus the three policies are non-preemptive. This proves part (i).
(ii) Since $\phi \in \Phi$ is non-location based, the queue in an RV subregion is $\Sigma GI/GI/1$ by Theorem 2. Since $\Sigma GI$ is a subset of $G$, then a $\Sigma GI/GI/1$ queue is also a $G/GI/1$ queue. Also, $\phi$ is non-preemptive and deadline smooth, Thus Theorem 1 of [8] holds in each RV subregion under $\phi \in \Phi$.

We define $R_{(m_j)} = T_{\text{dead}}(m_j) - T_{(m_j)}(m_j) - W_R(m_j)$, where $W_R(m_j)$ is the waiting time of Task$_{(m_j)}$, and is defined as the time difference between the arrival time $T_{(m_j)}^a$ and when $RV_m$ begins to travel to Task$_{(m_j)}$. Thus

$$T_{(m_j)}^{\text{sys}} = W_R(m_j) + T_{(m_j)}^{\text{reserv}}.$$  

Thus $\max\{T_{(m_j)}^{\text{sys}} - T_{(m_j)}^{\text{sys}}, 0\} = -\min\{T_{(m_j)}^{\text{sys}} - T_{(m_j)}^a, 0\} = -\min\{T_{(m_j)}^{\text{sys}} - T_{(m_j)}^a - W_R(m_j) - T_{(m_j)}^{\text{reserv}}, 0\}$.

When $\lambda^R < \mu^R$, the $\Sigma GI/GI/1$ queue at RV$_m$ is stable, we have $R_{(m_j)} \rightarrow R_{(m)} = T_{(m)}^{\text{dead}} - T_{(m)}^a - W_R(m)$ in distribution. Since all the RV subregions are homogeneous, we can write the generic term $R = T_{(m)}^{\text{dead}} - T_{(m)}^a - W_R(m)$.

Thus $E[T_{(m_j)}^{\text{sys}} - T_{(m_j)}^{\text{sys}}, 0] = E[-\min\{R - T_{(m_j)}^{\text{reserv}}, 0\}] = \int_{x=0}^{\infty} E[-\min\{R - t, 0\}] dF_{T_{(m_j)}^{\text{reserv}}}(t),$ where $F_{T_{(m_j)}^{\text{reserv}}}(t)$ is the cumulative distribution function (cdf) of $T_{(m_j)}^{\text{reserv}}$. Since function $g(x) = -\min\{x - t, 0\}$ is a convex function when $t$ is a constant, and $R$ has the same definition as $R$ in Theorem 1 of [8], then $E[-\min\{R^\phi - t, 0\}] \leq E[-\min\{R^\psi - t, 0\}]$ when $\phi \ll \psi$ for any constant $t$ by Theorem 1 of [8]. Thus $E[\max\{T_{(m_j)}^{\text{sys}} - T_{(m_j)}^{\text{sys}}, 0\}]^\phi \leq E[\max\{T_{(m_j)}^{\text{sys}} - T_{(m_j)}^{\text{sys}}, 0\}]^\psi$ when $\phi \ll \psi$, where the superscript $\phi$ means the value is obtained when the scheduling policy is $\phi$.

We know $TD = \frac{E[\max\{T_{(m_j)}^{\text{sys}} - T_{(m_j)}^{\text{sys}}, 0\}]}{E[T_{(m_j)}^{\text{sys}}]}$ by (17). Notice that $E[T_{(m_j)}^{\text{sys}}]$ is a constant, then $TD^\phi \leq TD^\psi$ when $\phi \ll \psi$. In particular, $TD^{EVDF} \leq TD^\phi$ since $EVDF \ll \phi$ for any $\phi \in \Phi$. Thus $TD^{EVDF} = \min_{\phi \in \Phi} TD^\phi$.

A.4. Proof of Theorem 4

Different task arrival processes will generate different tardiness values. We identify a worst-case arrival process maximizing tardiness. We prove the special case $\eta = 1$ of Definition 7 generates the worst case. This is Theorem 4.

**Definition 7.** An arrival process $\{T_{ki}^a\}_{i=1}^{\infty}$ is called an $\eta$-arrival process if

$$T_{ki}^a = \begin{cases} 0, & i \leq \eta \\ T_{ki}^{\text{dead}}, & i > \eta \end{cases}$$  

where $\eta \in \mathbb{N}$, and $T_{ki}^{\text{dead}}$ is given in (4).

For $\eta = 1$ the definition implies the arrival of the current task is the virtual deadline of the previous task. Thus the service times are also the interarrival times and the process at the output of the VV is identical to the arrival process, i.e., it is also an $\eta = 1$ process. Theorem 4 establishes the special role of the $\eta = 1$ process. It says all tardiness numbers in simulation are worst case.

**Theorem 4** Assume homogeneous virtual vehicles and homogeneous real vehicle subregions under the EVDF scheduling policy, let $\kappa \leq \kappa^c$, then the $\eta$-arrival process with $\eta = 1$ for all the virtual vehicles achieves the maximum $TD$ among all the renewal processes.

**Proof** To prove the $\eta$-arrival process with $\eta = 1$ maximizes $TD$, we show for any given task locations $\{X_{ki}\}$ and task sizes $\{T_{ki}^a\}$, an arbitrary renewal arrival process $\{T_{ki}^a\}$ will have a $TD$ less than that of the $\eta = 1$ arrival process. We denote by $\{T_{ki}^{a(\eta=1)}\}$ the $\eta = 1$ arrival process. Thus $T_{ki}^{a(\eta=1)} = T_{ki}^{\text{dead}(\eta=1)}$ by
Definition 7. For an arbitrary renewal arrival process \( \{T^a_k\} \), we construct an arrival process \( \{T^{a1}_k\} \) such that 
\[ T^{a1}_{ki} = \max \left\{ T^{a}_k, T^{a1}_k, T^{a1}(i+1) \right\} \]
for all \( k \) and \( i \). Thus \( T^{a1}_{ki} = \max \left\{ T^{a1}_k, T^{a1}(i+1) \right\} \). We construct another arrival process \( \{T^{s2}_k\} \) such that 
\[ T^{s2}_{ki} = \max \left\{ T^{s1}_k, T^{s1}(i+1) \right\}. \]
Note the three processes are constructed to have the same \( \{X_k\} \) and \( \{T^{s1}_k\} \). With the same \( \{X_k\} \) and \( \{T^{s1}_k\} \), we denote by \( TD^\phi (\{T^a_k\}) \) the tardiness under policy \( \phi \) when the arrival processes of the \( VV_k \) are \( \{T^a_k\} \). We want to show \( TD^{EVDF} (\{T^a_k\}) \leq TD^{EVDF} (\{T^{s2}_k\}) \) and 
\[ TD^{EVDF} (\{T^{s2}_k\}) \leq TD^{EVDF} (\{T^{a1}_k\}) \cdot \]

Since \( \{X_k\} \) and \( \{T^{s1}_k\} \) are the same for all three processes, then the service times \( T^{s1}_k \) are the same for the processes. For an arbitrary arrival process \( \{T^a_k\} \), we have \( T^{s2}_k \geq T^{s1}_k \) by \( (4) \) and Definition 7.

Comparing \( \{T^a_k\} \) and \( \{T^{a1}_k\} \), we have \( T^{a1}_k \leq T^{a1}_k \), and \( T^{a1}_k \leq T^{a1}(i+1) \) for all \( k \) and \( i \). The latter can be seen by induction on \( (4) \). First, \( T^{a1}_k \) and \( T^{a1}(i+1) \) imply \( T^{a1}_k \leq T^{a1}(i+1) \).

Under the arrival process \( \{T^{s2}_k\} \), we have 
\[ T^{s2}_k = \max \left\{ T^{s1}_k, T^{s1}(i+1) \right\} \]
and \( T^{s2}_k = \max \left\{ T^{s1}_k, T^{s1}(i+1) \right\} \). By \( (4) \), \( T^{s2}_k \) is always a subset of the set of tasks available to \( RV_m \) at any time since \( T^{s2}_k \leq T^{s1}_k \) for all \( k \) and \( i \). Thus \( TD^\phi (\{T^{s1}_k\}) \leq TD^{EVDF} (\{T^{s2}_k\}) \).

When every \( VV \) is under \( \{T^{a1}_k\} \), the interarrival time is \( T^{s1}_k \). The arrival process of the subregion of \( RV_m \), \( \{T^{a1}_k\} \), is the superposition of \( K \) thinned renewal processes of \( \{T^{a1}_k\} \) by Theorem 2. When every \( VV \) is under \( \{T^{s2}_k\} \), the interarrival time is \( T^{s2}_k \), where \( I^{a1}_k \) is the idle time between two consecutive tasks. Thus the arrival process of the subregion of \( RV_m \), \( \{T^{s2}_k\} \), is the superposition of \( K \) thinned renewal processes of \( \{T^{s2}_k\} \). Thus the generic interarrival time of \( \{T^{s2}_k\} \), \( I^{s2}_k \), is stochastically greater than that of \( \{T^{a1}_k\} \), \( I^{a1}_k \), i.e., \( I^{s2}_k \) is stochastically greater than \( B \), denoted \( A \geq tB \), if \( P(A > t) \geq P(B > t) \) for all \( -\infty < t < \infty \). Thus, we can construct a scheduling policy \( \phi \in \Phi \) in the subregion of \( RV_m \) under \( \{T^{s2}_k\} \) such that under \( \phi \), \( RV_m \) executes tasks in the same order as EVDF but always idles for \( I^{a1}_k - I^{a1}(k+1) \) right after the completion of each task. Since the virtual system time of each task is the same for both cases, Thus \( TD^\phi (\{T^{s2}_k\}) = TD^{EVDF} (\{T^{s2}_k\}) \).
$TD^{EVDF} (\{T_{ki}^2\}) \leq TD^{\phi} (\{T_{ki}^2\})$ by Theorem 3. Thus $TD^{EVDF} (\{T_{ki}^2\}) \leq TD^{EVDF} (\{T_{ki}^{(\eta=1)}\})$ for any given $\{X_{ki}\}$ and $\{T_{ki}^S\}$.

So we have $TD^{EVDF} (\{T_{ki}\}) \leq TD^{EVDF} (\{T_{ki}^{(\eta=1)}\})$ for an arbitrary renewal arrival process $\{T_{ki}\}$ for any given $\{X_{ki}\}$ and $\{T_{ki}^S\}$. Thus this is also true when we take expectation on $\{X_{ki}\}$ and $\{T_{ki}^S\}$. So the $\eta$-arrival process with $\eta = 1$ achieves the maximum $TD$ among all the renewal processes.

References