CYBER-PHYSICAL CLOUD COMPUTING LAB
UNIVERSITY OF CALIFORNIA, BERKELEY

SERVICE PROVISIONING IN CYBER-PHYSICAL CLOUD COMPUTING

Jiangchuan Huang, Clemens Krainer, Christoph M. Kirsch and Raja Sengupta

Working Papers
CPCC-WP-2012-11-01

CPCC Berkeley

November 2012

Service Provisioning in Cyber-Physical Cloud Computing *

Jiangchuan Huang
Systems, Civil and Environmental Engineering
University of California, Berkeley
jiangchuan@berkeley.edu

Clemens Krainer
Department of Computer Sciences
University of Salzburg, Austria
ckrainer@cs.uni-salzburg.at

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

Raja Sengupta
Systems, Civil and Environmental Engineering
University of California, Berkeley
sengupta@ce.berkeley.edu

ABSTRACT

We take the paradigm of cloud computing and apply it in a cyber-physical system where the servers (vehicles) can move in space and carry sensors and actuators. We call this cyber-physical cloud computing (CPCC). Like regular cloud computing, CPCC customers get a virtual machine running on a real server. The virtual machine is called a *virtual vehicle*. The CPCC server is called a real vehicle. We define virtual vehicles through the idea of a *virtual vehicle monitor* analogous to a virtual machine monitor. We design a service level agreement between cloud providers and customers. Resource-level and service-level metrics, and conformable applications are discussed through the concepts of virtual speed, required speed, delivered speed and token bucket regulator. We show that the mean required speed is bounded by the virtual speed and the "burstiness" is bounded by the bucket capacity. We prove that the mean delivered speed is greater than the mean required speed for an infinitely long computation sequence when the system is stable. The opposite is true when the system is not stable. We create a CPCC simulator implementing first come first served and gated traveling salesman policies. We show that a CPCC provider can host a given number of VVs with fewer RVs operating at the same speed while providing service guarantees on mean delivered speed and delivery probability.

*This work was partly supported by NSF-CNS-1136141.

1. INTRODUCTION

The time of cloud computing has finally come [2]. Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [20].

Major players such as Amazon Web Services, IBM SmartCloud, Windows Azure and Google App Engine have already made cloud computing a reality. Armbrust [2] summarized three new aspects in cloud computing from a hardware point of view: (i) The illusion of infinite computing resources available on demand; (ii) The elimination of an up-front commitment by cloud users; (iii) The ability to pay for use of computing resources on a short-term basis as needed.

When computation happens not only in time but also in space, the computation can be carried out by servers moving in space with a computing unit. Cloud computing that happens both in time and space is called cyber-physical cloud computing (CPCC) [3, 11]. The key innovations in CPCC [3] are to have servers (vehicles) move in space and carry sensors and/or actuators [11]. Examples of moving servers (vehicles) include a fleet of unmanned aerial vehicles (UAVs) equipped with cameras or sensors gathering data [26] or monitoring environment [22], in-vehicle smartphones as internet-connected sensor in road condition monitoring and alert application [7], and people carrying a smart phone in cloud sourcing [15].

Like regular cloud computing, CPCC customers get a virtual machine (VM) running on a real server. The VM is called a virtual vehicle (VV). The CPCC server is called a real vehicle (RV). In [11] we defined two kinds of mobility for a VV, *cyber-mobility*, a small-time-scale

hop, and *physical mobility*, a larger-time-scale motion with the RV. When a VV is bound to an RV, the VV exhibits physical mobility. When it migrates it exhibits cyber-mobility. CPCC, just like regular cloud computing, is meant to work at scale, i.e., at least for hundreds of real vehicles, and potentially thousands of virtual vehicles.

Cloud computing today is still in the early stages of development [31]. Current research mainly focus on cloud computing architecture [27], scheduling [9], resource allocation and load balancing [19], and resource provisioning and economic models [6]. Quality of service (QoS) can be found in [17, 5, 8]. Typically, the QoS is specified in a contract, a service level agreement (SLA), between the cloud provider and customer. Goiri et al. [8] characterizes the QoS by resource-level metrics (e.g., number and frequency of processors, FLOPS, etc.) and service-level metrics (e.g., service execution deadline). Current cloud computing providers mainly provide the resource-level metrics, e.g., Amazon EC2 standard Instances. The service-level metrics is only in availability, measured in annual uptime percentage. There is no guarantee on the system time, or how long does it take to complete the computation. Lee [16] concludes that the computing time can mainly be estimated in three ways: code analysis [24], analytic benchmarking/code profiling [30], and statistical prediction [10]. Akioka et al. [1] provided high performance computing benchmarks on Amazon EC2.

The telecommunication network service provisioning has a longer history than cloud computing. Kurose [14] gave a survey of QoS provisioning. The QoS guarantees are divided into *deterministic* guarantees (e.g., a bound on packets delay or loss) and *statistical* guarantees (e.g., no more than a specified fraction of packets will see performance below a certain specified value) in [29]. In ATM Networks, Low et al. [18] provided the deterministic QoS guarantee through regulating traffic at network edges using token bucket regulator. Token bucket or leaky bucket regulators [25, 4] are used to check compliance with contracts between users and network service providers.

Illuminated by the service provisioning experiences of cloud computing and telecommunication network, we believe a good CPCC service should have three aspects in its SLA, briefly summarized in Table 1.

- QoS resource-level metrics: the virtual vehicle specification by a constant: the virtual speed, v^V. Just as in cloud computing the customers reserve a virtual machine specified by number and frequency of processors, memory and storage, e.g., the Amazon EC2 standard Instances, in cyber-physical cloud computing, the customers reserve a virtual vehicle specified by the virtual speed v^V.
- Conformability: it defines the conformable applications that the customers can choose and check that they are providing demand requests conforming to the specification of the conformable applications. By the conformability definition, the cloud provider can guarantee that a customer does not generate the computations too fast, or does not generate large-size and long-distance-apart computations in a short period of time. We proposed to use a token bucket regulator to achieve this. The token bucket regulator provides a way to infer the resource requirements of a computation demand. The details of this regulation is in Section 3.
- QoS service-level metrics: We define the required speed by the customer, v_i^R, in Section 3.2, and define the delivered speed from the system, v_i^D, in Section 4.1 for each computation i. The QoS service-level metrics are the mean delivered speed for a computation sequence of length n, $\overline{v_i^{Dn}}$, and the probability that the delivered speed is greater than the required speed per computation, $P\left(v_i^D \geq v_i^R\right)$. The system guarantees that the mean delivered speed is greater than the mean required speed for a computation sequence of length n, and provides $P\left(v_i^D \geq v_i^R\right)$.

Table 1: CPCC service level agreement

Resource-level metrics	Virtual speed, v^V, Burstiness, c.
Service-level metrics	Guarantee that $\overline{v^{Dn}} > \overline{v^{Rn}}$, Provide $P\left(v_i^D \geq v_i^R\right)$.

We use token bucket regulators to ensure conformable application. Once an RV begins to approach a computation location it can not be interrupted until the RV has reached the location and completed the computation. This is different to cloud computing, where providers may migrate VMs arbitrarily between the nodes, computations may be suspended arbitrarily and resumed later [20]. So we regulate the computations at their arrival using a token bucket regulator, instead of controlling them at runtime.

Our contributions are as follows, we define virtual vehicle through the notion of virtual vehicle monitor following the formal definition of virtual machine [21] in Section 2. We also define the stability of the CPCC system in this section. We define conformable applications using a token bucket regulator in Section 3, and define the required speed, v_i^R, and its harmonic mean, $\overline{v^{Rn}}$. We show that the harmonic mean required speed is upper bounded by the virtual speed and the "bursti-

ness" of the required speed is upper bounded by the bucket capacity c in Theorem 1. We define the delivered speed, v_i^D, and its harmonic mean, $\overline{v^{Dn}}$, in Section 4. The delivered performance is measured by $\overline{v^{Dn}}$ and the probability that the delivered speed is greater than the required speed for a computation. We show in Theorem 2 that $\overline{v^{Dn}} \geq \overline{v^{Rn}}$ when the computation sequence length n goes to infinity when the system is stable, and $\overline{v^{Dn}} < \overline{v^{Rn}}$ when n is large enough when the system is not stable. Section 5 introduces the simulator implementing the concepts in Sections 2, 3, and 4, and shows the simulation results in Figures 4 and 5. Simulation shows that under G-TSP, 100 RVs with speed 1 m/s can host 200 VVs with virtual speed 1 m/s while the mean computation size is 0.25 s, and host 400 VVs with speed 1 m/s while the computation size is 0. The mean delivered speed is greater than $v^V = 1\ m/s$ for all the mean required speed range allowed by the SLA. The delivery probability per computation is greater than 90% for most range of the mean required speed. The provider can host a given number of VVs with fewer RVs operating at the same speed while providing service guarantees on mean delivered speed and delivery probability.

2. THE CPCC SYSTEM

A CPCC system is composed of $K \in \mathbb{N}$ independent virtual vehicles (VVs) and the infrastructure that hosts the K VVs in a convex region $\mathbf{A}$ of area A. The infrastructure includes $M \in \mathbb{N}$ real vehicles (RVs), and an *allocator* that allocates the RVs (resource) to the VVs (demands). The CPCC system is shown in Figure 1.

2.1 Real Vehicle

The infrastructure of the CPCC system is composed of M real vehicles (RVs) and the allocator. We assume that the allocator has central information of the system and has full control on the RVs subject to vehicle dynamics. Usually the vehicle speed is not a constant considering acceleration and deceleration, yaw constraint, or the affection of the environment. Examples of the dynamics of unmanned aerial vehicles (UAVs) can be found in [23] and the dynamics of cars can be found in [28]. We assume the j-th RV travels at constant speed v_j, $j = 1, \ldots, M$ when defining the concepts in Sections 3 and 4. We focus on homogeneous system where the dynamics of each RV are the same. We thus omit j in the following context.

2.2 Virtual Vehicle

The CPCC system provides K virtual Vehicles (VVs). The k-th VV is defined by a constant virtual speed, v_k^V, $k = 1, \ldots, K$, in the SLA between customer and CPCC provider.

A VV can be reserved by a customer to host a collection of computations in time and space that must be ordered into a sequence to be executed, subject to constraints that certain computations must be performed earlier than others. We represent the collection of computations hosted by the k-th VV as a *directed acyclic graph* (DAG), $\left(\mathbf{C}^k, \mathbf{E}^k\right)$, where $\mathbf{C}^k = \left\{C_i^k\right\}_{i=1}^n, n \in \mathbb{N}$ is the vertex set with each vertex, C_i^k, representing each computation, and $\mathbf{E}^k$ is the edge set with each edge representing each constraint. Algorithms for topological ordering may be used to generate a *valid* sequence of computations that satisfies the constrains in $\mathbf{E}^k$.

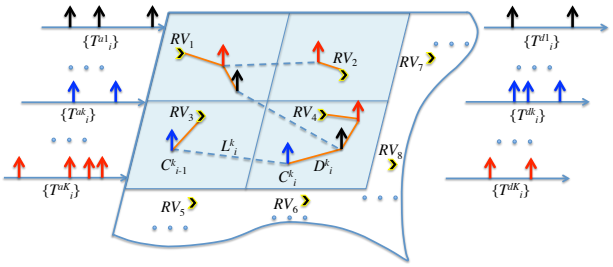

Figure 1: CPCC system.

Relabel $\left\{C_i^k\right\}_{i=1}^n$ such that it is a valid sequence, $C_1^k \to C_2^k \to \ldots \to C_n^k$. Then $C_i^k = \left(T_i^{ak}, X_i^k, T_{Ci}^k\right)$ is the i-th computation executed by the k-th VV, $i = 1, \ldots, n$, where T_i^{ak}, X_i^k and T_{Ci}^k are the arrival time (point), location requirement and computing time (interval) of C_i^k. When the system is stable, each computation will eventually be completed and departs the system. Let T_i^{dk} be the departure time (point) of the i-th computation. In this article we focus on homogeneous system where these values are i.i.d. in k. We thus omit k in the following context.

Let $F_X(.)$ be the experienced distribution of all the X_i's, and $f_X(.)$ be the pdf. Define

$$L_i = \| X_i - X_{i-1} \| \tag{1}$$

where $\| \cdot \|$ is the Euclidean norm defined on region $\mathbf{A}$, $i = 1, 2, \ldots$. $X_0^k = X_v$, which is the initial position of the RV allocated to the k-th VV, assume X_v is a random variable with distribution $F_X(.)$.

To the customers, the a VV is just like an RV. Following the definitions of virtual machine and virtual machine monitor by Popek et al. [21]. A virtual vehicle (VV) is taken to be an *efficient*, *isolated duplicate* of the real vehicle. We define virtual vehicles through the idea of a *virtual vehicle monitor* (VVM). The behavior of a VV is not exactly the same as an RV. We will show in Section 4 that a VV is not limited to only one RV at a time, but may execute computations on several RVs in parallel and in different places. This behavior means that a VV is something more capable than an RV.

2.3 Virtual Vehicle Monitor

An allocator is a *control program* that allocates the RVs (resources) to the computations (demands) hosted by the VVs. When a computation arrives, the allocator determines which RV the computation goes to. Following the definition of virtual machine monitor [21], the virtual vehicle monitor (VVM) can be defined as follows:

DEFINITION 1. *A virtual vehicle monitor (VVM) is any control program that satisfies the three properties of efficiency, resource control, and equivalence.*

DEFINITION 2. *A virtual vehicle (VV) is the environment created by the virtual vehicle monitor (VVM).*

One can check that the allocator satisfies these properties and thus the allocator is the VVM. Additionally, the environment created by the VVM provided to the customer is a VV.

The CPCC system is shown in Figure 1. The arrows in the same color represent the computations from the same VV. L_i^k is the distance between C_i^k and C_{i-1}^k, D_i^k is distance between C_i^k and the computation executed prior to C_i^k by the same RV. Under the VVM implementing certain policies, each vehicle will execute a set of computations possibly from all the VVs. When C_i arrives, it first waits for an RV to be allocated to it (wait time), then it waits for the vehicle to travel to it (travel time), then it spends some time under execution (execution time), where the execution time includes the migration time and computing time. Define the notations as follows:

- T_i is the system time of C_i, defined as the difference between the arrival of C_i, T_i^a, and the time C_i is completed, T_i^d. T_i is composed of the waiting time, T_{Wi}, and the service time, T_{Si}.
- T_{Wi} is the waiting time of C_i, defined as the difference between T_i^a and the time when an RV becomes available and starts to serve C_i.
- T_{Si} is the service time, defined as the difference between the time when an RV starts to serve C_i and T_i^d. T_{Si} is composed of the travel time, T_{Di}, the migration time, T_{Mi}, and the computing time, T_{Ci}.
- T_{Di} is the travel time, defined as the time interval between when the RV is available and when the RV reaches C_i. T_{Di} is determined by D_i as shown in Figure 1, and the dynamics of the RV executing C_i. $T_{Di} = \frac{D_i}{v}$ when we assume the RV has constant speed v.
- A migration time, T_{Mi}, is incurred if C_i is executed on a vehicle different from that of C_{i-1}. So $T_{Mi} > 0$ if a migration happens, and $T_{Mi} = 0$ otherwise.

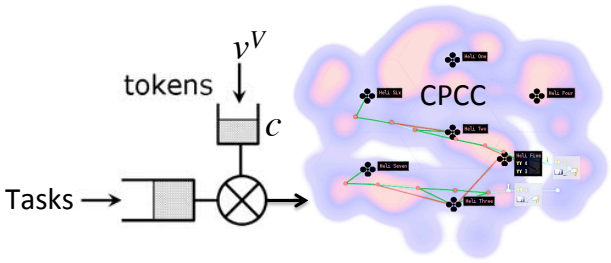

Figure 2: Token bucket regulator.

The relations are summarized in equation (2).

$$\begin{cases} T_i = T_i^d - T_i^a = T_{Wi} + T_{Si} \\ T_{Si} = T_{Di} + T_{Mi} + T_{Ci} \end{cases} \tag{2}$$

DEFINITION 3. *The CPCC system is said to be stable if $T_i \to T$ in distribution and $E[T]$ is finite.*

3. CONFORMABILITY

Different customers have individual time, location, and computation size requirements. Ideally, the CPCC system should allow all kinds of requirements. But the requirements cannot be delivered if there are too many requirements in a short period of time. We apply a token bucket regulator on the requirement arrivals, which gives large freedom on computation arrivals while making high guarantee on QoS service-level metrics possible.

3.1 Token Bucket Regulator

We use a token bucket regulator as shown in Figure 2 to regulate the computation arrivals that can be hosted on each VV. Define

$$\gamma_i = L_i + v^V T_{Ci} \tag{3}$$

$i = 1, 2, \ldots$, assume $T_0^a = 0$.

The bucket is empty at time 0. Tokens are added to the bucket at rate v^V. Each token represents 1 meter that the customer can use the VV to travel. The computing cost of T_{Ci} is also converted to the equivalent cost of meters through v^V. The bucket can hold at most c tokens. Denote by $\gamma(t)$ the amount of tokens in the bucket at time t, $\gamma(0) = 0$. The inflow tokens are discarded when the bucket is full, thus $\gamma(t) \leq c$. When C_i arrives, calculate γ_i, if $\gamma_i \leq \gamma(T_i^a)$, γ_i tokens are taken from the bucket, and C_i is sent to the cyber-physical cloud. If $\gamma_i > \gamma(T_i^a)$, no tokens are taken from the bucket, and C_i is considered to be *non-conformant*. Non-conformant computations are not allowed by the token bucket regulator. It is the customers' responsibility to provide conformant computations. In practice, non-conformant computations are simply be discarded. Please notice that here the amount of tokens is in real numbers.

Let $\gamma\left(T_i^{a+}\right)$ denote the amount of tokens in the bucket right after the arrival of the i-th computation, we have:

$$\gamma\left(T_i^{a+}\right) = \begin{cases} \gamma\left(T_i^a\right) - \gamma_i, & \text{if } \gamma_i \le \gamma\left(T_i^a\right) \\ \gamma\left(T_i^a\right), & \text{if } \gamma_i > \gamma\left(T_i^a\right) \end{cases}$$
$$\gamma(t) = \min\left\{c, \gamma\left(T_{i-1}^{a+}\right) + v^V\left(t - T_{i-1}^{a+}\right)\right\}, \quad T_{i-1}^a < t \le T_i^a \tag{4}$$

where $i = 1, 2, \ldots$, $T_0^{a+} = 0$, $\gamma(0) = 0$.

Under the same v^V and c, T_{Ci} and L_i are substitutable. The customers can generate computing intensive tasks with large T_{Ci} and small L_i, or generate traveling intensive tasks with small T_{Ci} and large L_i.

The larger v^V is, the larger or more frequent the computations can be. The larger c is, the more variant the computation arrivals can be. The CPCC provider can provide different values of v^V and c with different prices that the customers can choose from based on their own needs, just like the Amazon EC2 provides different instances with different computing capacities and different prices.

3.2 Required Speed from Customer

Define the *required speed* of C_i by the customer:

$$v_i^R = \frac{L_i}{T_i^a - T_{i-1}^a - T_{Ci}} \tag{5}$$

where $T_0^a = 0$ and $L_1 = \| X_1 - X_v \|$, where X_v is the initial vehicle position.

Define the mean required speed for a computation sequence of length n:

$$\overline{v^{Rn}} = \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_i^a - T_{i-1}^a - T_{Ci}\right)} = \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \frac{L_i}{v_i^R}} \tag{6}$$

$\overline{v^{Rn}}$ is the weighted harmonic mean of v_i^R, where the weights are L_i.

THEOREM 1. *A computation sequence of length n regulated by the token bucket regulator satisfies*

$$\overline{v^{Rn}} \le v^V \tag{7}$$

$$\max_{1 \le n_1 \le n_2 \le n} \sum_{i=n_1}^{n_2} \left(v_i^R - v^V\right)\left(T_i^a - T_{i-1}^a - T_{Ci}\right) \le c \tag{8}$$

PROOF. Under the token bucket regulator, starting with zero initial credits, the number of credits used by computation arrivals is less than the amount of credit arrived in the same time, i.e., outflow of credit $\le$ inflow of credit. For a computation sequence of length n,
$\sum_{i=1}^n L_i + v^V \sum_{i=1}^n T_{Ci} \le v^V \sum_{i=1}^n \left(T_i^a - T_{i-1}^a\right)$,
$i = 1, 2, \ldots$, assume $T_0^a = 0$.
Thus we have $\overline{v^{Rn}} \le \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_i^a - T_{i-1}^a - T_{Ci}\right)} \le v^V$

Since the capacity of the token bucket is c, any comformant subsequence of computations $\{C_i\}_{i=n_1}^{n_2}$, $n_1 \le n_2$, will not use more than c plus the arrival of credits in the time interval $\left(T_{n_1-1}^a, T_{n_2}^a\right]$.
$\sum_{i=n_1}^{n_2} L_i + v^V \sum_{i=n_1}^{n_2} T_{Ci} \le v^V \left(T_{n_2}^a - T_{n_1-1}^a\right) + c$
$= v^V \sum_{i=n_1}^{n_2} \left(T_i^a - T_{i-1}^a\right) + c$, $\forall 1 \le n_1 \le n_2 \le n$.
Plug in (5), we have
$\max_{1 \le n_1 \le n_2 \le n} \sum_{i=n_1}^{n_2} \left(v_i^R - v^V\right)\left(T_i^a - T_{i-1}^a - T_{Ci}\right) \le c$

□

Remark: The left hand side of (8) is called the maximum "burstiness" of the computation sequence, $\{C_i\}_{i=1}^n$.

DEFINITION 4. *A computation sequence of length n, $\{C_i\}_{i=1}^n$, is said to be compliant if (7) and (8) hold.*

The conformability definition says that the mean required speed of a customer is upper bounded by the virtual speed in the SLA. And the maximum "burstiness" of the computation sequence is upper bounded by the token bucket capacity, i.e., the required speed of a customer can be greater than the virtual speed in the SLA, but cannot be so long such that (8) is violated.

4. DELIVERED PERFORMANCE

A customer does not know the policy used by the allocator, or the arrival processes of other customers, so a customer does not know T_{Wi}, T_{Di} and T_{Mi} for each C_i. A customer considers the CPCC system to be a "black box" and only observes the arrival time, T_i^a, computing time, T_{Ci}, location, X_i and departure time, T_i^d, of C_i. A customer believes that T_i^d is determined by the location and the departure time of C_{i-1}, current job information (T_i^a, T_{Ci}, X_i), and the revealed (delivered) speed of the VV, v_i^D.

4.1 Delivered Speed from Provider

Define the *virtual service time*, T_{Si}^V, as the "service time" of C_i derived by the customer based on the information of X_{i-1}, T_{i-1}^d and $\left(T_i^a, T_{Ci}, X_i, T_i^d\right)$. Since $T_i^a \le T_i^d$, there are two cases:

- $T_{i-1}^d < T_i^a$, i.e., C_i arrives after the completion of C_{i-1}. Then the virtual vehicle is available when C_i arrives, it serves C_i from T_i^a to T_i^d, $T_{Si}^V = T_i^d - T_i^a$, which is the system of C_i.
- $T_{i-1}^d \ge T_i^a$, i.e., C_i arrives before the completion time of C_{i-1}. Then C_i needs to wait until C_{i-1} is completed and the virtual vehicle becomes available. The VV serves C_i from T_{i-1}^d to T_i^d, $T_{Si}^V = T_i^d - T_{i-1}^d$, which is the *inter-departure* time of C_i.

Thus we have

$$T_{Si}^V = \min\left\{T_i^d - T_i^a, T_i^d - T_{i-1}^d\right\} \tag{9}$$

$i = 1, 2, \ldots$ and $T_0^d = 0$.

Please notice that the virual service time, T_{Si}^V is not necessarily the real service time, T_{Si}, of C_i as shown in (2). Let v_i^D be the *delivered speed*, we have

$$T_{Si}^V = \frac{L_i}{v_i^D} + T_{Ci} \tag{10}$$

The delivered speed is

$$v_i^D = \frac{L_i}{T_{Si}^V - T_{Ci}} \tag{11}$$

Let T_i^c be the time when an RV starts executing C_i, i.e., $T_i^c = T_i^d - T_{Ci}$, then $T_i^a \le T_i^c \le T_i^d$. When $T_{i-1}^d < T_i^a$, $v_i^D > 0$, when $T_{i-1}^d \ge T_i^a$, there are two cases:

- $T_{i-1}^d \le T_i^c$, i.e., an RV starts to compute C_i after the completion of C_{i-1}, $v_i^D > 0$.
- $T_{i-1}^d > T_i^c$, i.e., another RV starts to compute C_i before the completion of C_{i-1}. C_{i-1} and C_i are executed in parallel, this is not allowed in sequential computing when we use only one RV to host the VV, but it is possible when more than one RVs can execute the computations hosted by the VV. The customer will have the illusion that the VV suddenly generates a copy at time T_i^c and location X_i, and starts excuting C_i.

Define the mean delivered speed for a computation sequence of length n:

$$\overline{v^{Dn}} = \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_{Si}^V - T_{Ci}\right)} = \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \frac{L_i}{v_i^D}} \tag{12}$$

$\overline{v^{Dn}}$ is the weighted harmonic mean of v_i^D, where the weights are L_i.

THEOREM 2. *When the CPCC system is stable,* $\lim_{n\to\infty}\left(\overline{v^{Dn}} - \overline{v^{Rn}}\right) \ge 0$. *When the CPCC system is not stable,* $\exists N > 0$, *s.t.* $n > N$ *implies* $\overline{v^{Dn}} - \overline{v^{Rn}} < 0$.

PROOF. When the system is stable, each of L_i, $T_i^a - T_{i-1}^a - T_{Ci}$, $T_i^d - T_i^a - T_{Ci}$ and $T_i^d - T_{i-1}^d - T_{Ci}$ are identically distributed and converge in distribution. Thus $T_{Si}^V - T_{Ci} = \min\left\{T_i^d - T_i^a, T_i^d - T_{i-1}^d\right\} - T_{Ci}$ are identically distributed and converge in distribution.
So $\lim_{n\to\infty} \frac{\sum_{i=1}^\infty L_i}{n}$, $\lim_{n\to\infty} \frac{\sum_{i=1}^\infty \left(T_i^a - T_{i-1}^a - T_{Ci}\right)}{n}$ and $\lim_{n\to\infty} \frac{\sum_{i=1}^\infty \left(T_{Si}^V - T_{Ci}\right)}{n}$ exist, where $\lim_{n\to\infty} Y_n$ exists means that Y_n converges in distribution.
So $\lim_{n\to\infty} \overline{v^{Rn}} = \lim_{n\to\infty} \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_i^a - T_{i-1}^a - T_{Ci}\right)}$
$= \frac{\lim_{n\to\infty} \frac{\sum_{i=1}^\infty L_i}{n}}{\lim_{n\to\infty} \frac{\sum_{i=1}^\infty \left(T_i^a - T_{i-1}^a - T_{Ci}\right)}{n}}$ exists. This limit does not go to infinity because $\overline{v^{Rn}} \le v^V$ by Theorem 1
And $\lim_{n\to\infty} \overline{v^{Dn}} = \lim_{n\to\infty} \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_{Si}^V - T_{Ci}\right)}$
$= \frac{\lim_{n\to\infty} \frac{\sum_{i=1}^\infty L_i}{n}}{\lim_{n\to\infty} \frac{\sum_{i=1}^\infty \left(T_{Si}^V - T_{Ci}\right)}{n}}$ exists. If this limit goes to infinity, then $\lim_{n\to\infty}\left(\overline{v^{Dn}} - \overline{v^{Rn}}\right) \ge 0$, so we assume that this limit is finite.
$\overline{v^{Dn}} = \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_{Si}^V - T_{Ci}\right)} \ge \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_i^d - T_{i-1}^d - T_{Ci}\right)}$
$\overline{v^{Dn}} - \overline{v^{Rn}} \ge \frac{\sum_{i=1}^n L_i \left(T_n^a - T_0^a - T_n^d + T_0^d\right)}{\sum_{i=1}^n \left(T_i^a - T_{i-1}^a - T_{Ci}\right) \sum_{i=1}^n \left(T_i^d - T_{i-1}^d - T_{Ci}\right)}$
$= \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_i^d - T_{i-1}^d - T_{Ci}\right)} \frac{\left(T_n^a - T_n^d\right)}{\sum_{i=1}^n \left(T_i^a - T_{i-1}^a - T_{Ci}\right)}$.
There are two cases:
If $\lim_{n\to\infty} \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_i^d - T_{i-1}^d - T_{Ci}\right)} = \infty$, then $\lim_{n\to\infty} \overline{v^{Dn}} \ge \lim_{n\to\infty} \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_i^d - T_{i-1}^d - T_{Ci}\right)} = \infty$. then $\lim_{n\to\infty}\left(\overline{v^{Dn}} - \overline{v^{Rn}}\right) \ge 0$ since $\overline{v^{Rn}} \le v^V$.
If $\lim_{n\to\infty} \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_i^d - T_{i-1}^d - T_{Ci}\right)}$ is finite,
then $\lim_{n\to\infty} \frac{\left(T_n^a - T_n^d\right)}{\sum_{i=1}^n \left(T_i^a - T_{i-1}^a - T_{Ci}\right)} = 0$,
because $\lim_{n\to\infty} \left(T_n^a - T_n^d\right) = -T$ is finite,
and $\lim_{n\to\infty} \sum_{i=1}^n \left(T_i^a - T_{i-1}^a - T_{Ci}\right) = \infty$.
Thus $\lim_{n\to\infty}\left(\overline{v^{Dn}} - \overline{v^{Rn}}\right) \ge$
$\lim_{n\to\infty} \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n \left(T_i^d - T_{i-1}^d - T_{Ci}\right)} \lim_{n\to\infty} \frac{\left(T_n^a - T_n^d\right)}{\sum_{i=1}^n \left(T_i^a - T_{i-1}^a - T_{Ci}\right)} = 0$

When the system is not stable, $\lim_{n\to\infty} \left(T_i^d - T_i^a\right) = \infty$, and $T_i^a - T_{i-1}^a$ is finite. So there $\exists N_1 > 0$, s.t., $i > N_1$ implies that $T_i^d - T_i^a > T_i^a - T_{i-1}^a$.
Let $M = \left|\sum_{i=1}^{N_1} \left(T_i^d - T_i^a - T_i^a + T_{i-1}^a\right)\right|$, then M is finite. So there $\exists N > N_1 > 0$, s.t., $i > N$ implies that $T_i^d - T_i^a > T_i^a - T_{i-1}^a + M$.
So $\sum_{i=1}^n \left(T_i^d - T_i^a\right) > \sum_{i=1}^n \left(T_i^a - T_{i-1}^a\right)$ when $n > N$.
Since $T_0^d = 0$ and $T_0^a = 0$, then $\sum_{i=1}^n \left(T_i^d - T_{i-1}^d\right) = T_n^d > T_n^a = \sum_{i=1}^n \left(T_i^a - T_{i-1}^a\right)$.
So $\sum_{i=1}^n \left(T_{Si}^V\right) = \sum_{i=1}^n \left(\min\left\{T_i^d - T_i^a, T_i^d - T_{i-1}^d\right\}\right) > \sum_{i=1}^n \left(T_i^a - T_{i-1}^a\right)$, then $\overline{v^{Rn}} > \overline{v^{Dn}}$, when $n > N$. □

5. SIMULATOR

The simulator demonstrates information-acquisition-as-a-service of mobile sensor networks for cyber-physical cloud computing (CPCC) as proposed in [3]. It offers fleets up to several dozens of helicopters and their sensors in such a way that participating applications do not recognize the difference between simulated and real flights. In addition, the simulator allows integration of external hardware for in-the-loop tests. Based on the JNavigator project [13] and the ESE CPCC class project [12] the implementation provides the simulation of physical helicopter swarms, the simulation of sensors, the virtual abstraction of autonomous vehicles (virtual vehicles), the migration of virtual vehicles among flying physical helicopters (real vehicles). Currently the sim-

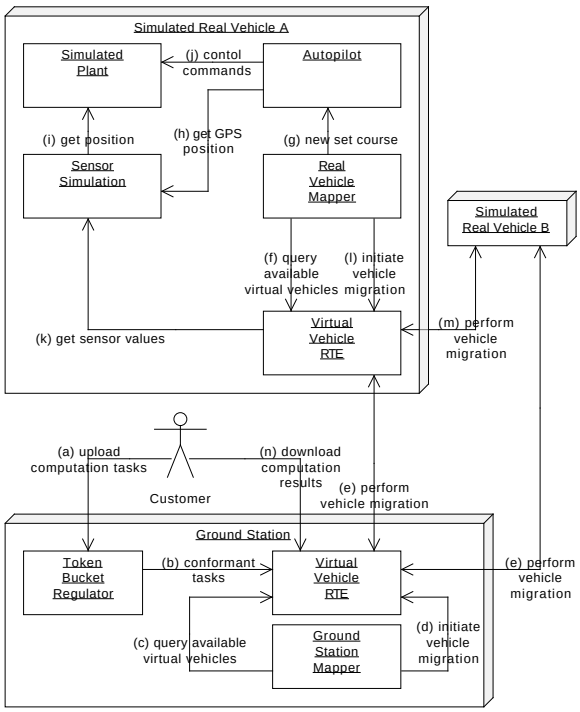

Figure 3: System overview

ulator supported sensor types are GPS receiver, photo camera, thermometer, barometer, and sonar.

Figure 3 provides an overview of the simulator system. It consists of a ground station (GS) and a number of simulated model helicopters, that is, simulated real vehicles (RVs). The GS contains the applications token bucket regulator (TBR), virtual vehicle run-time environment (VVRTE), and ground station mapper (GSM) that provide a web interface for uploading new computation tasks, downloading computation results, and administering VVs. A RV mainly comprises a model helicopter plant simulator with integrated flight control system, an autopilot, a sensor simulator, a real vehicle mapper (RVM), and a VVRTE.

The TBR ensures the conformability of computation arrivals, as described in section 3. It forwards conformant computation arrivals to the GS VVRTE and discards non-conformant computation arrivals. The GS VVRTE acts as container holding VVs with their state, i.e., their incomplete and completed tasks. The GSM decides which RVs should process incomplete VV tasks and commands the GS VVRTE to migrate the VVs accordingly. The RV VVRTEs execute VVs that perform information acquisition missions. VVs do not access RV sensors directly, but via the RV VVRTE. As commanded by the RVM, RV VVRTEs migrate VVs to other RV VVRTEs and the GS VVRTE. VV tasks that a RV can not process within its area of operations, cause the RVM to initiate migrations of the concerned VVs to adequate RVs. Based on the needs of unfinished VV tasks, the RVM calculates a set course and sends it to the autopilot for execution. After VVs have completed all their tasks, the RVM commands the migration back to the GS VVRTE.

The plant simulator emulates the helicopter's flight dynamics and inertial measurement unit. The integrated flight control system (FCS) operates attitude and altitude of the simulated vehicle. The autopilot stirs the simulated vehicle along a vehicle control language (VCL) script defined trajectory by issuing control commands to the FCS.

The flow of operation, as displayed in Figure 3, is as follows. A customer uploads new computation tasks to the TBR by using the GS web interface (a). The TBR verifies the conformability and forwards conformant computation arrivals to the GS VVRTE (b). The GSM queries the VVs available in the GS VVRTE having unfinished tasks (c) and initiates migrations to eligible RVs (d). The GS VVRTE migrates VVs to RVs as commanded by the GSM (e), e.g., to RV A. The RVM on RV A queries the RVs loaded in the RV VVRTE having unfinished tasks (f), calculates a new RV set course and sends it to the autopilot for execution (g). To stir the RV along the set course the autopilot retrieves the current RV position from the simulated GPS receiver (h) and issues control commands to the simulated plant (j). For the calculation of the RV's position in GPS coordinates, the simulated GPS receiver polls the current position of the simulated plant (i). After the RV has reached a VV task position, the concerned VV captures the required sensor values (k). If the current task of a VV is not in the flying range of the RV, the RVM initiates a migration to a suitable RV (l) and the RV VVRTE performs the ordered migration (m). Once all tasks of a VV have been completed, the RVM initiates a migration back to the GS VVRTE (l) and the RV VVRTE executes the requested migration (e). Now the customer may download the computation results from the GS VVRTE by using the GS web interface (n).

When performing a computation, a RV hovers at the required location. After completion, the RV proceeds to the next location to conduct the next computation. T_{Di} is the travel time between two locations, which is determined by vector D_i as shown in Figure 1, and the dynamics of the RV executing computation C_i. We require that the RV travels at an average speed v and calculate $T_{Di} = \frac{\|D_i\|}{v}$. Since the RV hovers when executing a computation, the RV has to accelerate and decelerate between two locations. The simulator applies equation 13 to stir the RV from location X_{i-1} to location X_i. $X_i(T)$ is the desired RV location at time

T, where $0 \leq T \leq T_{Di}$. The RV starts at time $T = 0$ and reaches X_i at time $T = T_{Di}$.

$$X_i(T) \quad = \quad X_{i-1} + D_i \frac{T^2}{T_{Di}^2} \cdot \left(3 - 2 \cdot \frac{T}{T_{Di}}\right) \qquad (13)$$

Currently, the RVM provides two policies for calculating new RV set courses: First Come First Served (FCFS) policy and Gated Traveling Salesman Policy (G-TSP). The policies consider only task positions within the according RV's range of operation and work as follows.

The FCFS policy causes a RV to fly to the position of the task having the earliest arrival time.

The G-TSP policy takes a snapshot of the positions of the active tasks of all currently available VVs in the RV's VVRTE. Then, it calculates a TSP tour starting at the current RV position, traversing all task positions, and ending at the RV's depot position. After that, the policy removes the depot position from the TSP tour and causes the RV to perform the TSP tour. Any tasks arriving during the tour are ignored for now. After the RV has finished the tour, the G-TSP policy takes a snapshot again and starts from the beginning.

Table 2: Simulation setup

Simulation Parameters	
Area Size	10 m × 10 m
Cell Size	1 m × 1 m
# VVs, K	200 and 400
# RVs, M	100
Virtual speed, v^V	1 m/s
RV speed	1 m/s
Token bucket capacity, c	4 m
Mean computing time, $E[T_{Ci}]$	0 and 0.25 s
Migration time, T_{Mi}	0
# Computations per VV	100
Policies	FCFS and G-TSP

The simulation setup is summarized in Table 2. The simulation was done on a square region of size 10 $m \times$ 10 m. There are 100 RVs, each travels at 1 m/s. The region was divided into 100 (10×10) square cells, each with size 1 $m \times 1m$. Assign one vehicle to each cell. Each vehicle serves its own cell independently under FCFS or G-TSP. The token bucket regulator has token arrival rate $v^V = 1\ m/s$ and bucket capacity $c = 4\ m$. The migration time of each computation, $T_{Mi} = 0$. Cases that the system hosts 200 and 400 VVs with virtual speed $v^V = 1m/s$, and uniformly distributed task size with mean 0 and 0.25 s are simulated and results are shown in Figure 4 and 5. Both figures give the results for the range of mean required speed up to $v^V = 1\ m/s$ as allowed in the SLA.

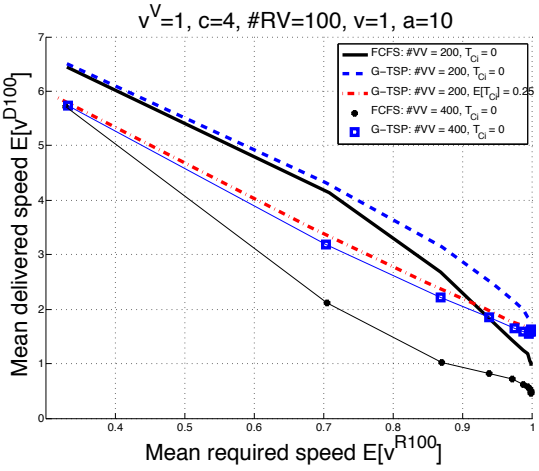

Figure 4: Mean delivered speed with increasing mean required speed.

Figure 4 shows that the mean delivered speed of a computation sequence of length 100, $\overline{v^{D100}}$ or $E\left[v^{D100}\right]$, decreases as the mean required speed, $\overline{v^{R100}}$ or $E\left[v^{R100}\right]$, increases. Under FCFS, when #VVs = 200 and computation size = 0, the mean delivered speed is greater than $v^V = 1$, but when #VVs = 400, the mean delivered speed falls below $v^V = 1$ when the mean required speed is greater than 0.9 m/s. Under G-TSP, the mean delivered speed is still greater than v^V when #VVs = 400 and computation size = 0. So G-TSP is a better policy. Increasing the number of VVs will decrease the mean delivered speed under the same mean required speed under FCFS and G-TSP.

Figure 5 shows that the delivery probability, $P\left(V_i^D > V_i^R\right)$, decreases as the mean required speed increases. Under G-TSP, when #VVs = 200 and $T_{Ci} = 0$, $P\left(V_i^D > V_i^R\right) \geq 0.9$ when the mean required is less than 0.9 m/s. But $P\left(V_i^D > V_i^R\right)$ falls quickly to 0.55 when the mean required speed approaches $v^V = 1$. The delivery probability is high for most range of mean delivered speed allowed by the SLA. Similar pattern follows for the cases when #VVs = 200, $E[T_{Ci}] = 0.25$ and #VVs = 400, $T_{Ci} = 0$ under G-TSP, where $P\left(V_i^D > V_i^R\right) \geq 0.9$ when the mean required is less than 0.8. To host the same #VVs with the same $E[T_{Ci}]$, FCFS has lower delivery probability than G-TSP which indicates that G-TSP is better. Higher #VVs and larger $E[T_{Ci}]$ results in lower $P\left(V_i^D > V_i^R\right)$ under the same mean required speed.

From the results we know that under G-TSP, 100 RVs with speed 1 m/s can host 200 VVs with virtual speed 1 m/s while the mean computation size is 0.25 s and host 400 VVs with speed 1 m/s while the computation size is 0. The mean delivered speed is greater than v^V = 1 m/s for all the mean required speed range allowed

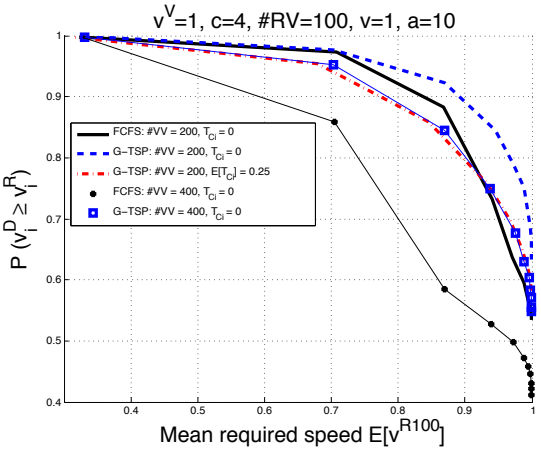

Figure 5: Delivery probability with increasing mean required speed.

by the SLA. The delivery probability per task is greater than 90% for most range of the mean required speed. The provider can host a given number of VVs with fewer RVs operating at the same speed while providing service guarantees on mean delivered speed and delivery probability.

6. CONCLUSION

This paper takes the concept of cyber-physical cloud computing (CPCC) introduced in [11] and designs a service provisioning model that makes CPCC as a service possible. We define the virtual vehicle through the idea of a *virtual vehicle monitor* (VVM) following the formal definition of Popek et al. [21]. We design a service level agreement (SLA) between the cloud provider and customer. The SLA includes resource-level metrics, conformability definitions, and service-level metrics. The resource-level metrics is a virtual speed, a constant provided by the CPCC provider. The conformable application is defined through a token bucket regulator. The service-level metrics include the mean delivered speed and the probability that the delivered speed is greater than the required speed for a computation. We show that the mean required speed is bounded by the virtual speed and the "burstiness" of the required speed is bounded by the bucket capacity in Theorem 1. We show in Theorem 2 that $\overline{v^{Dn}} \geq \overline{v^{Rn}}$ when the computation sequence length n goes to infinity when the system is stable, and $\overline{v^{Dn}} < \overline{v^{Rn}}$ when n is large enough when the system is not stable.

The virtual vehicle monitor, implementing FCFS and G-TSP policies, is simulated and the delivered performance is given in Figures 4 and 5. The simulation shows that the provider can host a given number of VVs with fewer RVs operating at the same speed while providing service guarantees on mean delivered speed and delivery probability.

7. REFERENCES

[1] S. Akioka and Y. Muraoka. HPC Benchmarks on Amazon EC2. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pages 1029 –1034, april 2010.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[3] S. Craciunas, A. Haas, C. Kirsch, H. Payer, H. Röck, A. Rottmann, A. Sokolova, R. Trummer, J. Love, and R. Sengupta. Information-Acquisition-as-a-Service for Cyber-Physical Cloud Computing. In *Proc. Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2010.

[4] R. L. Cruz. A calculus for network delay. Part I. Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, Jan. 1991.

[5] J. Fito, I. Goiri, and J. Guitart. SLA-driven Elastic Cloud Hosting Provider. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 111 –118, feb. 2010.

[6] A. Gera and C. H. Xia. Learning Curves and Stochastic Models for Pricing and Provisioning Cloud Computing Services. *Service Science*, 3(1):99–109, 2011.

[7] A. Ghose, P. Biswas, C. Bhaumik, M. Sharma, A. Pal, and A. Jha. Road condition monitoring and alert application: Using in-vehicle Smartphone as Internet-connected sensor. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 489 –491, march 2012.

[8] I. n. Goiri, F. Julià, J. O. Fitó, M. Macías, and J. Guitart. Resource-level QoS metric for CPU-based guarantees in cloud providers. In *Proceedings of the 7th international conference on Economics of grids, clouds, systems, and services*, GECON'10, pages 34–47, Berlin, Heidelberg, 2010. Springer-Verlag.

[9] Y.-C. Hsu, P. Liu, and J.-J. Wu. Job sequence scheduling for cloud computing. In *Cloud and Service Computing (CSC), 2011 International*

Conference on, pages 212 –219, dec. 2011.

[10] M. A. Iverson, F. Ozguner, and G. J. Follen. Run-Time Statistical Estimation of Task Execution Times for Heterogeneous Distributed Computing. In *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*, HPDC '96, pages 263–, Washington, DC, USA, 1996. IEEE Computer Society.

[11] C. Kirsch, E. Pereira, R. Sengupta, H. Chen, R. Hansen, J. Huang, F. Landolt, M. Lippautz, A. Rottmann, R. Swick, R. Trummer, and D. Vizzini. Cyber-Physical Cloud Computing: The Binding and Migration Problem. In *Design, Automation and Test in Europe*, 2012.

[12] M. Kleber, C. Krainer, A. Schröcker, and B. Zechmeister. ESE Cyber-Physical Cloud Computing Project class project for the Embedded Software Engineering Course, Winter 2011/2012.

[13] C. D. Krainer. JNavigator - An Autonomous Navigation System for the JAviator Quadrotor Helicopter. Master's thesis, University of Salzburg, Austria, 2009.

[14] J. Kurose. Open issues and challenges in providing quality of service guarantees in high-speed networks. *SIGCOMM Comput. Commun. Rev.*, 23(1):6–15, Jan. 1993.

[15] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 48(9):140 –150, sept. 2010.

[16] G. Lee. *Resource Allocation and Scheduling in Heterogeneous Cloud Environments*. PhD thesis, EECS Department, University of California, Berkeley, May 2012.

[17] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini. SLA-Driven Clustering of QoS-Aware Application Servers. *IEEE Trans. Softw. Eng.*, 33(3):186–197, Mar. 2007.

[18] S. Low and P. Varaiya. A New Approach to Service Provisioning in ATM Networks. *IEEE/ACM Transactions on Networking*, 1:547–553, 1993.

[19] S. T. Maguluri, R. Srikant, and L. Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In A. G. Greenberg and K. Sohraby, editors, *INFOCOM*, pages 702–710. IEEE, 2012.

[20] P. Mell and T. Grance. The NIST Definition of Cloud Computing, Version 15, Oct 2009.

[21] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[22] S. Rathinam, Z. Kim, A. Soghikian, and R. Sengupta. Vision Based Following of Locally Linear Structures using an Unmanned Aerial Vehicle. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pages 6085–6090, 2005.

[23] S. Rathinam, R. Sengupta, and S. Darbha. A Resource Allocation Algorithm for Multivehicle Systems With Nonholonomic Constraints. *Automation Science and Engineering, IEEE Transactions on*, 4(1):98 –104, jan. 2007.

[24] B. Reistad and D. K. Gifford. Static dependent costs for estimating execution time. *SIGPLAN Lisp Pointers*, VII(3):65–78, July 1994.

[25] J. Roberts and C. . P. M. Committee. *Performance Evaluation and Design of Multiservice Networks: COST 224 : Final Report, October 1991*. EUR (Series). Commission of the European Communities, Directorate-General Telecommunications, Information Industries, and Innovation, 1992.

[26] A. Ryan and J. Hedrick. A mode-switching path planner for UAV-assisted search and rescue. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pages 1471 – 1476, dec. 2005.

[27] W.-T. Tsai, X. Sun, and J. Balasooriya. Service-Oriented Cloud Computing Architecture. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 684 –689, april 2010.

[28] G. Venture, P.-J. Ripert, W. Khalil, M. Gautier, and P. Bodson. Modeling and Identification of Passenger Car Dynamics Using Robotics Formalism. *Intelligent Transportation Systems, IEEE Transactions on*, 7(3):349 –359, sept. 2006.

[29] D. Verma, H. Zhang, and D. Ferrari. Guaranteeing delay jitter bounds in packet switching networks. In *Proc. Tricomm*, pages 35–43, Chapel Hill, North Carolina, 1991.

[30] J. Yang, I. Ahmad, and A. Ghafoor. Estimation of Execution times on Heterogeneous Supercomputer Architectures. In *Proceedings of the 1993 International Conference on Parallel Processing - Volume 01*, ICPP '93, pages 219–226, Washington, DC, USA, 1993. IEEE Computer Society.

[31] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18, 2010. 10.1007/s13174-010-0007-6.