Optimizing Vehicle-to-Cloud Data Transfers using Soft Real-Time Scheduling Concepts

Jean Ibarz, Michaël Lauer, Matthieu Roy, Jean-Charles Fabre, Olivier Flébus

Optimizing Vehicle-to-Cloud Data Transfers using Soft Real-Time Scheduling Concepts

Jean Ibarz[1,2,3], Michaël Lauer[1,4], Matthieu Roy[1,5], Jean-Charles Fabre[1,3], Olivier Flébus[2]

1. CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
2. Continental Digital Services France, 1 Avenue Paul Ourliac, F-31100 Toulouse, France
3. Univ de Toulouse, INP, F-31400 Toulouse, France
4. Univ de Toulouse, UPS, F-31400 Toulouse, France
5. Univ de Toulouse, LAAS, F-31400 Toulouse, France

ABSTRACT

The main promise of intelligent transportation systems (ITS) is that leveraging the information sensed by millions of vehicles will increase the quality of the user's experience. However, the unpredictable nature of road events, combined with a projected network overload, calls for a careful optimization of the vehicles' data transfers, taking into account spatio-temporal, safety and value constraints. In this article, we provide a methodical solution to optimize vehicle-to-cloud (V2C) data transfers, based on a series of steps. First, we show that this optimization problem can be modeled as a soft real-time scheduling problem. Second, we provide an extension of a classical algorithm for the generation of workloads, by increasing its coverage with regards to our use-case representation. Third, we estimate the bounds of an optimal clairvoyant algorithm in order to have a baseline for a fair comparison of existing scheduling algorithms. The results show that, within all these algorithms, one clearly outperforms the others regardless of the load rate. Interestingly, its performance gain increases when overload grows, and it can be implemented very efficiently, which makes it highly suitable for embedded systems.

KEYWORDS

distributed sensing, real-time, IoT, V2C, mobile system, embedded system, event-based, data collection, reliability

ACM Reference Format:
Jean Ibarz[1,2,3], Michaël Lauer[1,4], Matthieu Roy[1,5], Jean-Charles Fabre[1,3], Olivier Flébus[2]. 2020. Optimizing Vehicle-to-Cloud Data Transfers using Soft Real-Time Scheduling Concepts. In *28th International Conference on Real-Time Networks and Systems (RTNS 2020), June 9–10, 2020, Paris, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394810.3394818

1 INTRODUCTION

Intelligent transportation systems (ITS) promise a dramatic increase of the safety and efficiency levels of vehicles while reducing CO_2 emissions, by harnessing the power of global and real-time information on traffic and road status [28]. Developing a reliable level of environment awareness at global scale requires that millions of connected vehicles act as crowdsourcers in a massive distributed system. For this purpose, each vehicle is equipped with several sensors to capture local information from the highly dynamic environment it is evolving in. All vehicles will generate events depending on their respective perception, and will send these events to the Cloud using vehicle-to-cloud (V2C) communications [14]. Such an aggregation paves the way for reliable and up-to-date knowledge of the environment, using big data mining.

However, Cisco [20] forecasts a tremendous increase of mobile accesses to the infrastructure, with mobile traffic speeds growing by roughly 60% —from 18 to 29Mbps—, while the mobile traffic is expected to increase by 170% —from 29 to 77 Exabytes per month—. Furthermore, the V2C data flow is expected to be so enormous that optimization will be critically important, both technically and economically. In order to maximize the usefulness of the data received in the Cloud, an adequate data prioritization scheme is essential to decide which data should be transmitted, and when.

Optimizing V2C transfers requires taking into account spatio-temporal constraints, due to the dynamics of connected vehicular systems. For example, if a vehicle is aware of a free parking spot in a highly prized parking lot, this information will rapidly lose its utility with aging, as the free parking space may be taken by anyone at any moment. On the contrary, information related to a speed limit traffic sign will likely be valid for days.

Using soft real-time systems concepts, here we propose an experimental evaluation of several scheduling algorithms, at the end of solving the V2C data flow optimization problem.

Contributions. To begin, we show how this problem can be translated into a value-based scheduling problem in the soft real-time theory. Then, we compare performances of existing on-line soft real-time scheduling algorithms, using an experimental approach. For this purpose, we extend an existing method for experimental evaluation of scheduling algorithms, in order to improve the coverage of its analysis. Experiments are evaluated against two instances of an optimal clairvoyant scheduling algorithm that provide bounds for evaluation and a fair baseline for comparison [5, 7, 18]. The obtained results indicate that the Dynamic Timeliness Deadline algorithm (DTD1) is the most efficient one in terms of utility performance, and is a promising candidate to handle the V2C data transfer optimization problem.

RTNS 2020, June 9–10, 2020, Paris, France

ACM ISBN 978-1-4503-7593-1/20/06…$15.00
https://doi.org/10.1145/3394810.3394818

2 CONTEXT AND PROBLEM STATEMENT

Connected vehicles generate *events* from the raw information captured by sensors in a highly dynamic environment. For example, a connected vehicle may generate an event to signal the presence (or absence) of a traffic sign, or an event to signal an instance of emergency braking. Due to the highly dynamic nature of vehicles and the environment, future events are hardly predictable.

At the logical level, events are contained in *messages*; a *message* may be composed of multiple events.

At the network level, messages are split into fixed size *packets*. We assume that sending a packet is an atomic action that cannot be preempted. However, it is possible to suspend (resp. resume) the transmission of a message by halting (resp. starting) the transmission of the remaining packets. We assume that preempting the transmission of a message is instantaneous.

Messages are transmitted to the Cloud individually by each vehicle via a unique communication channel in a sequential manner, i.e. one message at a time. The available communication bandwidth is known and constant during the simulation length. Despite these are very strong assumptions in our context, where real-life network's available bandwidth may vary and be hard to predict, the relevance of our experimental results is justified by the choice of evaluated algorithms. Evaluated algorithms take decisions very frequently. In fact, no more than one packet is send after each decision taken by an algorithm. Therefore, the algorithms have the capability to rapidly adjust to unexpected changes, e.g. an abrupt lowering of the available bandwidth or the failure in transmission of a message. Furthermore, an algorithm schedules only one message to send at a time. Because the decisions are very short sighted in the time horizon, unexpected changes should have minor impact on the algorithms' performances.

When received by the Cloud, a message generates some *utility* to the global system. An exact definition of *utility* is out of the scope of this paper; but roughly speaking, it represents the contribution of every message to the quality of service provided to end users. The generated *utility* value is clearly related to the content of the message, and also to temporal properties associated to each of the events it contains. Let us consider an event containing information related to the mobility of a vehicle. After a few seconds, the vehicle may have moved several hundreds of meters or may have changed its direction, making the event irrelevant for safety-critical purposes such as collision avoidance. On the contrary, an event indicating the presence of a traffic sign may still be useful for the global system hours or days after its generation.

Traffic quality of service (QoS) in wireless communication networks has been subject to considerable research efforts in recent years. Meeting QoS requirements in a resource-constrained environment, such as a sensor network, is exceptionally challenging [1, 15, 32]. In a highly mobile sensor network such as vehicular ones, QoS challenges are even higher. It is also impractical to bound the load of V2C data flows, because network dynamics may cause havoc in an emergency data transfer [3]. Accounting for such considerations, we assume that not being able to send an event on time does not have catastrophic consequences on the system.

It is expected that a huge amount of data will be generated by millions of vehicles. Sending all the events generated by each vehicle is not an option, since mobile access infrastructure is already at its capacity limit today. Anyway, in the eventuality that such solution would be technically feasible, the amount of data aggregated in the Cloud would incur unsustainable storage and processing costs. Hence, it must be decided which message must be sent (first) and when, considering temporal constraints, in order to maximize the utility acquired by the global system under limited resources.

We assume isolation between vehicles, meaning that vehicles do not cooperate with each other. Hence, the problem of maximizing the utility acquired by the global system is reduced to the problem of maximizing the utility for each vehicle considered individually. This assumption allows us to considerably reduce the complexity of the problem for better scalability.

In this paper, we do not consider how messages are generated from events, and assume that arrival dates of future messages cannot be known in advance. Consequently, we focus on *dynamic* (*on-line*, *on-the-fly*) scheduling algorithms for unknown and overloaded situations.

The problem we tackled through our work here is finding an efficient dynamic scheduling for sending a set of messages (that have utility and temporal properties), in a way that the resulting utility is maximized for each vehicle considered individually. In the next Section, we show that this problem can be mapped as a soft real-time scheduling problem.

3 MODELING

3.1 Soft real-time scheduling problem

Let us consider a real-time scheduling problem where a set of N *independent* jobs have to be run on a single processor. Each job has an arrival date and cannot be processed before its arrival. A job requires a fixed amount of *execution units* to be processed. Each unit of time, the processor can be either idle, or processing. Processor speed refers to the number of execution units that can be processed each *unit of time* (when not idle). At time k, if $s[k]$ denotes the processor speed and $\overline{c}_j[k]$ denotes the remaining execution units of the job being processed, $min\left(\overline{c}_j[k], s[k]\right)$ execution units are processed during that unit of time.

In our problem, the messages to be transmitted by the mobile network are similar to the jobs to be processed by the processor. Hence, the number of units of execution to process a job is directly related to the number of packets to transmit a message. The analogy between our scheduling problem and the real-time scheduling problem is synthesized in Table 1.

Table 1: Analogy of our problem and real-time scheduling

Our scheduling problem	A real-time scheduling problem
A message	A real-time job
Packets of a message	Execution units of a job
Communication interface	Processor
Available bandwidth	Processor's speed
Sending of a packet	Processing of a unit of execution
Transmission of a message	Processing of a job

Following the usual terminology [13], a real-time job is:

- *soft*, if producing the results after its deadline has still some utility for the system, although causing a performance degradation,
- *firm*, if producing the results after its deadline is useless for the system, but does not cause any damage,
- *hard*, if producing the results after its deadline may imply catastrophic consequences for the system under control.

If the usefulness of producing the results of a job is independent of its completion date, the job is said to be *non* real-time.

As we already discussed in Section 2, the transmission of a message is not safety-critical. Therefore, the sending of a message is either a non, a soft, or a firm real-time job. In line with the model introduced in [12], we characterize a real-time job J_i by a 5-tuple $< a_i, c_i, v_i, d_i, \delta_i >$, where:

- $a_i \in \mathbb{N}$ is the *arrival date* of the job. A job is not instantiated and cannot be processed before its arrival date. A non-clairvoyant algorithm is not aware of a job before its arrival date.
- $c_i \in \mathbb{N}, c_i > 0$ is the number of execution units required to be processed for the job to complete. It is an integer value, corresponding to the transmission of packet on the network, which we assumed is atomic.
- $v_i \in \mathbb{R}^+$ is the *base value* of the job.
- $d_i \in \mathbb{N}$ is the *(relative) firm deadline* of the job.
- $\delta_i \in \mathbb{N}$ is the *lateness limit* of the job.

Following the concept introduced by Douglas Jensen in [21], a function $\phi_i(k)$, referred to as the *value function* of job i, is associated to each job. This value function expresses the value that would be granted to the system upon completion of its associated job, as a function of the completion date. $\phi_i(k)$ is illustrated in Figure 1 and is defined by the following constraints:

$$\begin{cases} \phi_i(k) = v_i & \text{if } k \in [a_i; a_i + d_i] \\ \phi_i(k) = (k - d_i - a_i)\,\delta_i & \text{if } k \in]a_i + d_i; a_i + d_i + \delta_i] \\ \phi_i(k) = 0 & \text{otherwise.} \end{cases}$$

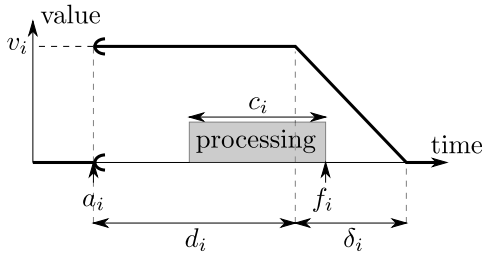


Figure 1: Utility function of a soft real-time job

Notice that, because $c_i > 0$, the value of a job at its activation date $\phi_i(a_i)$ is equal to its base value v_i. Notice also that restricting (a_i, c_i) to $\mathbb{N}^2$, combined with our fixed processor execution speed assumption, allows us to run discrete-time simulations with a fixed time-step. Hence, it greatly simplifies the implementation of the simulator, which in turn reduces the risk of implementation errors. The assumption that $(d_i, \delta_i) \in \mathbb{N}^2$ simplifies reasoning without sacrificing expressivity, as real numbers would not increase the power of the model.

From this formal definition of a soft real-time job and utility function, let us derive the following observations:

- $a_i + d_i$ is the *(absolute) firm deadline* of job i.
- $a_i + d_i + \delta_i$ is the *(absolute) soft deadline* of job i.
- The difference between a firm real-time job and a soft real-time job is the lateness limit parameter δ_i: $\delta_i > 0$ for a soft real-time job, and $\delta_i = 0$ for a firm real-time job.
- By convention, we choose that $d_i = \delta_i = +\infty$ for a non real-time job.

Because this model of a soft real-time job requires only five parameters, it is memory efficient and hence appropriate to be used in embedded systems.

Definition 1. A job is said to be *completed* if and only if all of its required execution units have been processed. We denote by f_i the completion date of job i.

Definition 2. The *lateness* of a job i is defined as $max(0, f_i - d_i)$ if the job is completed, undefined otherwise.

A completed job is said to be *late* if and only if its lateness value is (strictly) greater than 0. A real-time system is said to be *overloaded* when not all jobs can be completed on time.

In our context, events are generated aperiodically in an uncontrollable and unpredictable way. Furthermore, the available bandwidth can be as low as zero. In such conditions, system overload is very likely to occur and will be a *nominal case*.

3.2 Evaluation methodology

A way to evaluate a performance guarantee of an on-line scheduler is to compare it with a *clairvoyant* [26] scheduler, i.e. one that knows *a priori* all the parameters of all the jobs of the scheduling problem. Analytical approaches to evaluate the efficiency of an online scheduling algorithm generally involve the so-called *competitive ratio* metric, introduced by Sleator and Tarjan in [30].

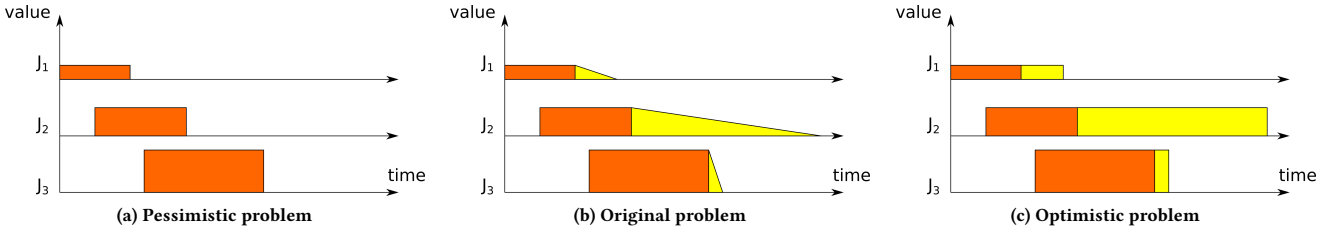
Definition 3. The *competitive ratio* is defined as

$$min\left(\left\{\frac{\Gamma_A(P)}{\Gamma_{OPT}(P)} | P \in \Omega\right\}\right)$$

where

- $\Gamma_A(P)$ is the cumulative value obtained by an algorithm A on the scheduling problem P,
- $\Gamma_{OPT}(P)$ is the cumulative value obtained by a clairvoyant optimal algorithm on the scheduling problem P,
- Ω is the set of all existing scheduling problems [11].

Baruah et al. have proved in [7] that the competitive ratio of an on-line scheduling algorithm is upper bounded by 1/4. However, this result is only representative of a made up worst case scenario. In this scenario, the overload has an arbitrary (but finite) duration, all jobs have zero laxity, the same value density (i.e. their value is equal to their computation time), and their execution time can be arbitrarily small [12]. Since in most real world applications job characteristics are much less restrictive, the 1/4 bound has only a theoretical validity [12, 31]. Hence, because the competitive ratio provides a performance guarantee that is not required in our context, an experimental approach seems more appropriate to fit more tightly real world conditions.



(a) Pessimistic problem (b) Original problem (c) Optimistic problem

Figure 2: Original soft real-time problem bounded by a pessimistic problem and an optimistic one.

Our goal is to schedule a set of jobs in order to maximize the overall value of the system, a scheduling paradigm known as *value-based scheduling* [10]. In this context, the most relevant metric is probably the ratio Γ_A/Γ_{OPT} between the *cumulative value* Γ_A obtained by an algorithm A and the cumulative value Γ_{OPT} obtained by an optimal clairvoyant algorithm OPT. However, the problem of finding an optimal scheduling is generally intractable from a computational perspective. Finding an optimal value-based scheduling for a given set of firm real-time jobs in a uniprocessor system is similar to finding a *schedulable* subset of jobs so that the aggregated value of completed jobs is maximal. A set of firm real-time jobs is said to be *schedulable* if and only if there exists a schedule such that all jobs complete by their deadline. This problem can be shown to be reducible from the knapsack problem [19], and hence is NP-Hard [8].

Definition 4. The *Hit Value Ratio* (*HVR*) is defined as the ratio Γ_A/Γ between the *cumulative value* $\Gamma_A \triangleq \sum_i \phi_i(f_i)$ obtained by an algorithm A and the *total value* $\Gamma \triangleq \sum_i v_i$ of the job set [11].

The HVR can be viewed as a heuristic metric of Γ_A/Γ_{OPT}, avoiding the calculation of an optimal schedule but still allowing a comparative evaluation of different algorithms in a meaningful dimension.

Other metrics such as job preemption, job lateness, job success ratio, etc., are not expected to have an important impact on our context, and therefore have been omitted.

In order to provide a better insight of the efficiency of evaluated algorithms, we propose to go deeper and estimate the performance of an optimal clairvoyant scheduler. When considering soft real-time jobs, the utility acquired by the system is the value returned by the non-linear utility function of the completed job. It follows that the cumulative value acquired by the system is also non-linear and the completion date of each job must be known. Hence, the problem of finding an optimal value-based scheduling for a set of soft real-time jobs is even tougher than the one for a set of firm real-time jobs, which is already NP-hard.

To alleviate this issue, we derive two simpler problems: a pessimistic problem and an optimistic problem, where the soft real-time jobs are transformed into firm real-time jobs. As shown in Figure 2, in the pessimistic problem, the soft deadline value of a job is replaced by its firm deadline value, that is a job grant no value if its firm deadline is not met. In the optimistic problem, the firm deadline value of a job is replaced by its soft deadline value, that is a job grant all of its value if its soft deadline is met, no value otherwise.

As stated before, despite being much easier than the original problem, these problems are nevertheless NP-Hard. These two bounding problems are formulated into MILP (Mixed Integer Linear Problem) in order to make use of powerful solvers that are nowadays available. For our experimentations, we use the open source available GLPK (Gnu Linear Programming Kit) solver.

Because all jobs are firm real-time ones, either a job j is completed without exceeding its deadline and the job grants a utility v_j to the overall system, either the job grants no value to the overall system because it does not complete at all or it completes too late. The problem of finding a schedule that maximizes the total cumulative value is equivalent to the problem of finding a schedulable subset of jobs that maximizes the total cumulative value. Also, it is well known that for any schedulable set of firm jobs, EDF algorithm is optimal.

Therefore, and since we previously assumed a constant execution speed of one, a set of jobs is schedulable if and only if it satisfies the following EDF exact schedulability test [16]:

Theorem 1. *A set of aperiodic jobs is schedulable (by EDF) if and only if*

$$\forall\,(i,k)\,,\, a_i < d_k, \quad \sum_{j:a_i \le a_j \text{ and } d_j \le d_k} c_j \le d_k - a_i$$

From this, a MILP formulation was directly derived:

maximize

$$\sum_i y_i v_i$$

subject to

$$\forall\,(i,k)\,,\, a_i < d_k, \quad \sum_{j:a_i \le a_j \text{ and } d_j \le d_k} \left(y_j c_j \le d_k - a_i\right)$$

where y_i is a boolean decision variable, indicating whether the job i belongs to the schedulable subset or not, and where a_i, d_i, c_i, v_i are the parameters of the job as defined in 3.1.

When a solution to this problem is found, that means we found a schedulable subset of jobs that maximizes the hit value ratio. Therefore, a valid scheduling can be obtained by applying an EDF algorithm that ignores all jobs that do not belong to the schedulable subset of jobs. This is a convenient way to use the simulator to generate results from this scheduling. Results obtained by simulation can then be advantageously used in order to check for implementation errors. For example, one can check that the total cumulative value obtained by simulation is equal to the aggregated value of all the jobs that belong to the schedulable subset of jobs, or check that all jobs that belong to this subset are completed with no lateness, i.e. without exceeding their firm deadline.

In spite of these simplifications, the problem is still too complex in some cases. Finding an optimal solution in a reasonable amount of time is not always possible. The solver was then used with the following parameters:

(1) "−−pcost": branch using hybrid pseudocost heuristic,
(2) "−−mipgap 0.02": set relative gap tolerance to 2%, that is we stop when a pseudo-optimal solution is found which is less than 2% distant from the optimal solution.

Using these parameters, a solution was found in a few seconds most of the time, and it took up to 15 minutes for the hardest instances[1].

3.3 Scenarios generation

We want to generate a set of scenarios in order to produce experimental results which will allow some statistical analysis. It was chosen to generate a set of scenarios with random jobs characteristics by using the method proposed in [2], slightly adapted for the purpose of increasing the evaluation coverage. In [2], the characteristics of jobs are pseudo-randomly generated following some arbitrary distribution laws. As a result, generated scenarios may not be representative of real use cases. Therefore, we propose to generate jobs' characteristics pseudo-randomly according to a set of arbitrary distribution laws, that are themselves chosen randomly. This concept, illustrated in Figure 3, increases the diversity of generated scenarios, and hence the evaluation coverage.

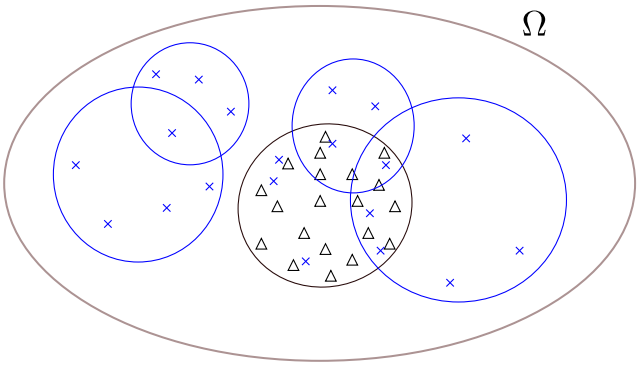


Figure 3: Ω represents the set of all possible scenarios. Disks represent classes of scenario. Black color is state-of-the-art [2], blue is contribution. Black triangles are a possible execution of the state-of-the-art algorithm. Blue crosses represent a possible execution of our proposed algorithm.

A scenario is a set of jobs that needs to be processed. For each scenario, jobs characteristics are pseudo-randomly generated according to Algorithm 1.

The algorithm contains two main parts:

(1) lines 4 to 7 (scenario class): a class of scenario is randomly chosen, by picking the functions that determine how the jobs characteristics will be generated. This block constitutes the main introduced modifications, compared to the method used in [2].

[1]Using GLPK solver v4.65 on a computer equipped with a processor Intel Core i5-6440HQ and 2x4 Go of SDRAM DDR4-2133 running Lubuntu 64 bits.

input :
- $N \in \mathbb{N}^*$: the number of jobs,
- σ: the nominal load rate.

output:
- J: a set of N soft real-time jobs.

1 $J \leftarrow \emptyset$
2 $f_a \leftarrow X_a \sim EXP\left(\frac{\sigma}{E(X_c)}\right)$
3 *Choose a class of scenario*
4 $f_c \leftarrow X_c \sim U(1,100)$ or $X_c \sim LU(1,100)$
5 $f_v \leftarrow Id(f_c)$ or $Inv(f_c)$ or $X_v \sim U(1,100)$ or $X_v \sim LU(1,100)$
6 $f_d \leftarrow X_d \sim U(1,10)$ or $X_d \sim U(1,200)$ or $X_d \sim U(100,200)$ or $X_d \sim LU(1,10)$ or $X_d \sim LU(1,200)$ or $X_d \sim LU(100,200)$
7 $f_\delta \leftarrow X_\delta \sim U(1,10)$ or $X_\delta \sim U(1,200)$ or $X_\delta \sim U(100,200)$ or $X_\delta \sim LU(1,10)$ or $X_\delta \sim LU(1,200)$ or $X_\delta \sim LU(100,200)$
8 *Generate N jobs in the class*
9 $a_0 \leftarrow 0$
10 **for** $i \leftarrow 1$ **to** N **do**
11 $\quad a_i \leftarrow Round\left(\sum_{j=1}^{j=i} f_a(j)\right)$
12 $\quad c_i \leftarrow Round\left(f_c(i)\right)$
13 $\quad v_i \leftarrow f_v(i)$
14 $\quad d_i \leftarrow Round\left(c_i + f_d(i)\right)$
15 $\quad \delta_i \leftarrow Round\left(f_\delta(i)\right)$
16 $\quad J_i \leftarrow < a_i, c_i, v_i, d_i, \delta_i >$
17 $\quad J \leftarrow J \cup J_i$
18 **end**

Algorithm 1: Generation of a scenario with N soft real-time jobs and a targeted nominal load rate λ.

(2) lines 9 to 17 (job generation): a set of N jobs is generated, according to the chosen class of scenarios, that will constitute a sampling for evaluation.

Algorithm 1 uses the following classical conventions:

- $Id()$: the *identity* function,
- $Inv()$: the *inverse* function,
- $E(X)$: the *mean* or *expected value* of a random variable X,
- $CDF(X)$: the *cumulative distribution function* of a random variable X,
- $X \sim U(min, max)$: a *uniform* random variable X, i.e. a random variable X such that

$$CDF(X) = \begin{cases} 0 & \text{if } X < min \\ \frac{X-min}{max-min} & \text{if } min \leq X \leq max \\ 1 & \text{if } X > max \end{cases}$$

- $X \sim LU(min, max)$: a *log-uniform* random variable X, i.e. a random variable X such that

$$CDF(X) = \frac{ln(X) - ln(min)}{ln(max) - ln(min)}$$

- $X \sim EXP(\lambda)$: an *exponential* random variable X, i.e. a random variable X such that

$$CDF(X) = 1 - e^{-\lambda X}$$

The *inverse transform sampling* is used to generate uniform, log-uniform, and exponential random variables:

- $U(min, max) \sim (max - min) \times U(0, 1) + min$,
- $LU(min, max) \sim e^{U(ln(min), ln(max))}$,
- $EXP(\lambda) \sim -ln\,(U(0, 1))\,/\lambda$.

Following classical hypotheses [2], inter-arrival dates of a given sequence of jobs are generated according to a random variable X_a that follows an exponential distribution $EXP(\lambda), \lambda \triangleq \frac{\sigma}{E(X_c)}$, σ being the desired load (line 2 in Algorithm 1). Activation date a_i is computed as the rounded value of aggregated inter-arrival dates of previous jobs, including the current one (line 11).

Notice that the variance of an exponential random variable is $E^2(X)$, while the variance of a uniform variable is $\frac{(b-a)^2}{12} = \frac{(2(E(X)-a))^2}{12}$. In our case, $a > 0$, hence the variance of a uniform variable is strictly less then $\frac{1}{3}E^2(X)$, i.e. strictly less than $1/3$ the variance of an exponential random variable. Therefore, sampling inter-arrival dates using a random exponential variable is more likely to cause *bursty* arrivals, i.e. that a large amount of jobs arrive within a small period of time, providing a more representative behavior of our ITS use case.

3.4 Simulation model

The simulator used in this work has been developed in Python. The simulation consists in executing the following steps in a loop until an arbitrary ending date is reached:

(1) Activate all jobs at current cycle k.
(2) Abort jobs that do not have a strictly positive utility value at current time.
(3) Get a scheduling decision and process it: if the scheduled job is not the current running job, then halt the current job (if any), and start or resume the scheduled job.
(4) Wait until next cycle $k+1$ and update all jobs characteristics accordingly (remaining units of execution, completed jobs, etc.).

4 RESULTS AND ANALYSIS

Simulations have been run with 1000 runs for each load rate in the set $\{25\%, 100\%, 400\%, 1600\%\}$. Each run consists of 100 jobs generated with Algorithm 1, that uses the following parameters:

- $f_c()$, associated to the jobs execution units, is chosen with equal probability to be either a uniform random variable in $[1; 100]$ or a log-uniform random variable in $[1; 100]$.
- $f_v()$, associated to the jobs value, is chosen with equal probability to be:
  - $Id(f_c)$, i.e. the job value is equal to its number of execution units. In this particular case, the best case value-density $v_i/c_i = 1$ of jobs is the same.
  - $Inv(f_c)$, i.e. the value of a job is equal to the inverse of its number of execution units. Contrarily to the previous case, where best case value-density was the same for all jobs, here the best case value-density $v_i/c_i = 1/c_i^2$ of jobs will be exponentially distributed.
  - A uniform random variable in the range $[1; 100]$.
  - A log-uniform random variable in the range $[1; 100]$.
- $f_d()$, associated to the jobs best case laxity $d_i - c_i$, is chosen with equal probability to be a uniform or a log-uniform random variable. Independently, the boundings are chosen with equal probability to be either $[1; 10]$ for tight only deadlines, $[100; 200]$ for loose only deadlines, or $[1; 200]$ for tight to loose deadlines.
- $f_\delta()$, associated to the jobs lateness limit, is chosen in the same manner as $f_d()$, but independently.

The set of evaluated algorithms is composed of *i*) Static Value Density (SVD), *ii*) Semi Dynamic Value Density (SDVD), *iii*) Dynamic Value Density (DVD1), *iv*) Dynamic Value Density Squared (DVD2), *v*) Dynamic Timeliness Deadline (DTD1), and *vi*) Dynamic Timeliness Deadline Squared (DTD2). All are greedy algorithms, and work as follow: at every time step, the algorithm calculates the heuristic score for all active jobs. Then, if the current job has the best heuristic score, its processing continue for the next time step. However, if another job has a strictly better heuristic score than the one of the current job, it preempts the current job. Each algorithm is based on its own heuristic and are described in Table 2. Notice that real-life network speeds will vary and hence be hardly predictable. Greedy algorithms have the advantage of being very "short-sighted" —their scheduling decisions involve only one job in the time horizon—, hence they should be less sensitive to external perturbations such as execution speed variations or jobs arrivals. In [11], SDVD is said to exhibit a very graceful degradation during overloads to enjoy a low sensitivity to jobs set parameters. SVD, DVD1 and DTD1 have been considered because of their resemblance to SDVD. Additionally, algorithms DVD2 and DTD2 have been chosen because, in [2], DVD2 was shown to outperform DVD1 while DTD2 was envisioned to perform better than DTD1.

Particular attention has been placed on deciding how to end a scenario simulation. While tedious and not our main contribution, correctly fixing the ending date of the simulation in order to maximize the representativeness of the generated results is mandatory and not trivial. One solution may be to run the simulation until the latest soft deadline. In this case, the obtained load rate may sensibly differs from the targeted load rate. For example, let's consider a scenario with $n = 100$ jobs, a targeted load rate $\sigma = 1600\%$, and all jobs having the same characteristics except for the activation date. Any job requires the processing of 10 execution units for completion and have a relative (relative to the job activation date) soft deadline of 90. Therefore, the activation date of jobs is given by an exponential random variable X_a with a fire rate $\lambda = \frac{\sigma}{E(X_c)} = 16/10 = 1.6$. The latest job activation is expected to be $n\lambda = 160$ and, because the relative soft deadline is 90 for any job, the simulation is expected to run for 250 units of time. However, the total number of execution units to be processed is equal to $10n = 1000$, resulting in an effective load rate of $1000/250 = 400\%$ which clearly doesn't match with 1600%, the targeted load rate. Another solution may be to run the simulation until the final date $F = n\lambda$. In this case, the average load rate obtained would be exactly the targeted load rate, as in average the last activation date would be $n\lambda$. However, some jobs might be infeasible, because their activation date would simply exceed the ending date, or because

Table 2: On-line scheduling algorithms evaluated in this paper. Color code: static parameter, dynamic parameter, dynamic parameter with foreseeing future.

Algorithm	Heuristic (job priority $\propto$ heuristic value)
SVD (Static Value Density)	v_i / c_i
SDVD (Semi Dynamic Value Density)	$\phi_i(t)$ / c_i
DVD1 (Dynamic Value Density)	$\phi_i(t)$ / $\overline{c}_i(t)$
DVD2 (Dynamic Value Density Squared)	$\phi_i(t)$ / $\overline{c}_i^2(t)$
DTD1 (Dynamic Timeliness Deadline)	$\phi_i\ (t + \overline{c}_i(t))$ / $\overline{c}_i(t)$
DTD2 (Dynamic Timeliness Deadline Squared)	$\phi_i\ (t + \overline{c}_i(t))$ / $\overline{c}_i^2(t)$

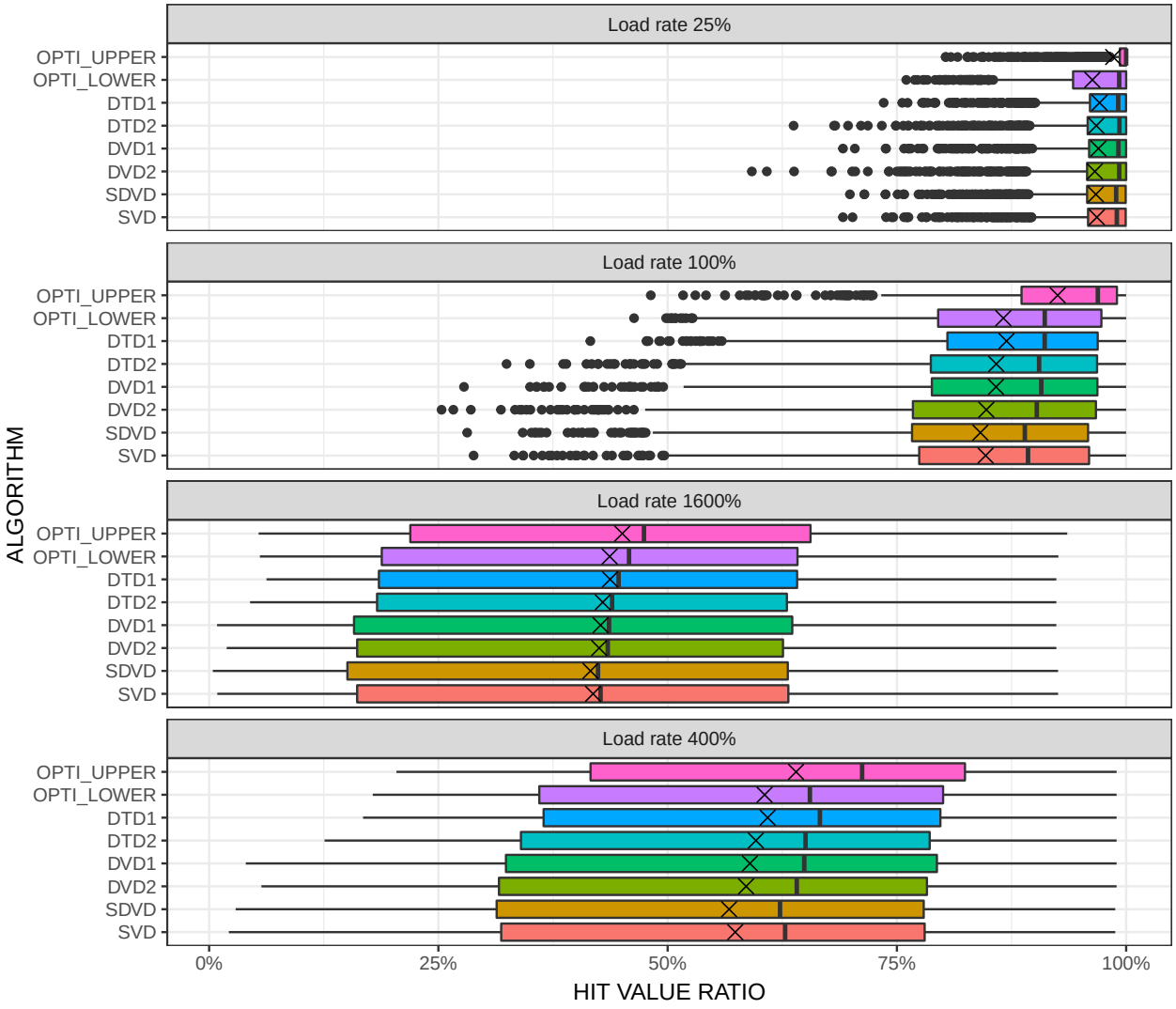


Figure 4: Hit Value Ratio measured on clairvoyant and online algorithms using 1000 scenarios for each load rate. OPTI_LOWER (resp. OPTI_UPPER) corresponds to the optimal clairvoyant algorithm, estimated with an at most 2% relative error, obtained from the firm real-time scheduling problem where jobs' deadlines equal the firm deadline (resp. soft deadline) of the original problem. The mean value is indicated by a cross. The lower and upper hinges (resp. the left bar and the right bar of a box) correspond to the first and third quartiles, i.e., 25$^{\text{th}}$ and 75$^{\text{th}}$ percentiles. The upper whisker extends from the hinge to the largest value no further than 1.5 × IQR from the hinge —the Inter-Quartile Range (IQR) is the distance between the first and third quartiles—. The lower whisker extends from the hinge to the smallest value at most 1.5 × IQR of the hinge. Outliers are plotted individually, as black disks.

they may be activated too late to be able to be completed before the simulation ending. To fix this issue, we propose to remove all infeasible jobs, i.e. jobs such that the sum of their activation date and their required units of execution exceed $n\lambda$. The consequence of this is that the obtained load rate will be lowered, as in the previous case, however with a minor effect: the ending date would be equal to $100 \times 1.6 = 160$, and infeasible jobs would be jobs activated after date $160 - 10 = 150$, i.e. approximately $10 \times 1.6 = 16$ jobs in average. The effective average load rate would then be lowered by 16% of the initally targeted load rate. Another issue raised by this simulation ending is that scheduling algorithms are not aware of the simulation interruption, and may take wrong decisions because of this. Some scheduling algorithms may be less affected then others, such that algorithms who prioritize the execution of smaller jobs. To deal with this, all deadlines exceeding the simulation ending date F are set to F. By doing so, we ensure that all algorithms are aware of the right amount of available time before the jobs lose all of their value, avoiding the introduction of a bias in the obtained results.

Figure 4 plots the Hit Value Ratio (HVR) performance of the chosen algorithms for each load rate in the set {25%, 100%, 400%, 1600%}.

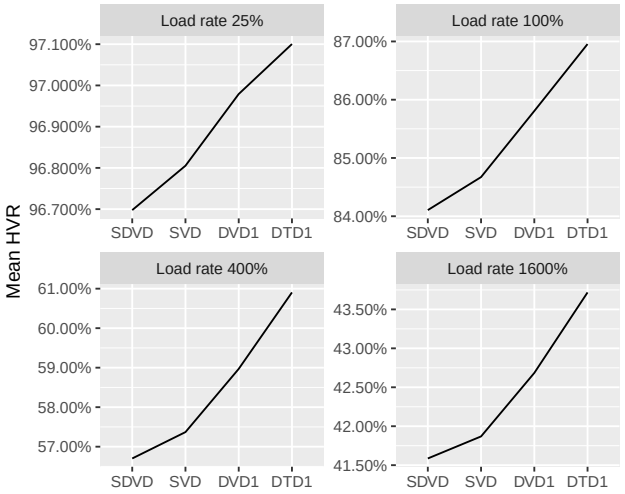


Figure 5: Mean Hit Value Ratio performed by different online algorithms over 1000 scenarios for each load rate.

The mean HVR for each algorithms under each load rate is plotted in Figure 5, and shows how algorithms can be sorted by efficiency, with respect to the mean HVR as a performance criteria, as SVD < DVD1 < DTD1. Furthermore, this ordering holds no matter the load rate. SDVD, despite the fact that its heuristic uses dynamic information (the current value of a job), is less efficient under all load rates than algorithm SVD which uses only static information. We explain this result by the fact that when a job is becoming more and more tardy, its value also decrease more and more and so does the SDVD heuristic score, improving the risk that the job become preempted by another one, no matter if its current value density is very high or not.

DVD1 and DTD1 heuristics, respectively $\phi_i(t)/\overline{c}_i$ and $\phi_i(t+\overline{c}_i)/\overline{c}_i$, where $\overline{c}_i$ represents the remaining number of execution units $c_i - c_i(t)$ at current time, only differ in the way the value of a job is calculated. In the case where all jobs are non real-time, i.e. $\forall i, d_i = +\infty$, $\phi_i()$ is a constant function and the two heuristics are equivalent. As a consequence, the two algorithms are equivalent and must perform equally. In another case where jobs must meet some deadlines, the tightest the time constraints, the highest may be the score difference between the two heuristics. One may then expect that the performance difference between the two algorithms increase with the tightness of jobs time constraints.

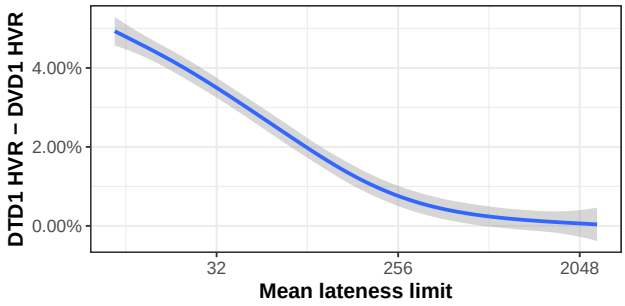


Figure 6: DTD1 HVR minus DVD1 HVR, smoothed using LOESS method, as a function of jobs mean lateness limit. 200 scenarios are generated for each lateness limit value in the set {0, 10, 20, 40, 80, 160, 320, 640, 1280, 2560}, with a load rate of 400% and according to algorithm 1. In each scenario, jobs parameters are modified such that for all i, $d_i = c_i$ and δ_i equals the chosen lateness limit value. Eventually, the solution discussed in Section 4 is applied to determine the ending date of the scenarios, to remove unfeasible jobs, and to adjust (if necessary) jobs parameters such that absolute firm and soft deadlines does not exceed the simulation duration.

In Figure 6, we evaluated the algorithms DVD1 and DTD1 by making vary the lateness limit. In each scenario, 100 of firm real-time jobs are generated and they all have the same lateness limit value. A set of 200 scenarios was generated and simulated for each possible lateness limit value in {0, 10, 20, 40, 80, 160, 320, 640, 1280, 2560}. As expected, results show that the lower is the lateness limit value of the jobs, the higher is the performance difference between the two algorithms. As in previous results, DTD1 performs better then DVD1, and our results indicate that this is true whatever the lateness limit value.

According to [2], DVD2 is expected to perform better then DVD1. However, the results shown in Figure 4 indicate an opposite trend: in average, DVD2 is less effective than DVD1, at all load rates. Besides, the statistical significance of the observed performance difference have been validated in Appendix A. The same observation can be done when comparing DTD2 to DTD1. Our explanation for this trend is that DVD2 and DTD2 may suffer from the sunk cost effect, which corresponds to a higher tendency to continue an endeavor once an investment in money, effort, or time has been made [4]. Let's consider a firm real-time job that has already been partly processed, and the scheduling algorithm can choose between the following two end results if it realizes that a job with a higher value-density $v_j/c_j > v_i/\overline{c}_i(t)$ and the same remaining number of execution units $c_j = \overline{c}_i(t)$ arrived:

(1) Having paid the price of the processed execution units of job i and having suffered from continuing processing job i, or;
(2) Having paid the price of the processed execution units of job i and having used the remaining time to process job j since its arrival.

When job j arrives, the cost associated to the partial processing of job i is a sunk cost because it has already been incurred and it cannot be recovered. As a consequence, this cost is not relevant to decisions about the future. Also, because $c_j = \overline{c}_i(t)$, the same amount of resources is required to complete either job i or job j. However, completion of job j produces a greater utility value $v_j > v_i$ to the system, so clearly here the rational choice is the second one. Heuristics of algorithms DVD2 and DTD2 tend to give higher priority to jobs about to complete, a consequence of the squaring of $\overline{c}_i(t)$. This is supposed to potentially avoid preemption of partially executed jobs, hence avoiding the wastage of unrecoverable resources spent earlier. As a result, they suffer from the sunk cost effect and make sometimes irrational choices, which results in a loss of effectiveness that is visible in our results. To give a better insight of how much harm the squaring of $\overline{c}_i(t)$ can produce, let's think of the following case. Let be two kind of jobs: short jobs and long jobs. Suppose that a small job has the following characteristics at its date of arrival: $< v_s, c_s >$, and a long job has the following characteristics: $< QKv_s, Kc_s >$, where $(Q, K) \in \mathbb{N}^2, Q > 1, K >> 1$. That is to say, a long job is K times longer to process than a small job, but the value density at arrival date of a long job is Q times higher than that of a small job. There is one small job arrival for each date in the set $\{n \times c_s | n \in N\}$, and one long job arrival for each date in the set $\{n \times c_l | n \in N\}$. Let's denote by h_s (resp. h_l) the heuristic score evaluated at date of arrival of a small job (resp. long job) for algorithm DVD2 (or DTD2). Clearly, processing K small jobs requires the same amount of time than processing one long job, because $c_l = K \times c_s$. However, processing a long job produces a Q times higher outcome than processing K small jobs, because $v_l = Q \times K \times v_s$. We have $h_s = v_s/c_s^2, h_l = (Q/K) \times (v_s/c_s^2)$. In this particular case, when $Q/K < 1$, i.e. when K is big enough compared to Q, the algorithm DVD2 (or DTD2) always process small jobs instead of long jobs. For example, if long jobs have a value density 10 times higher, it is enough to have long jobs more than 10 times longer than small jobs to get the wrong decision with DVD2 (or DTD2), that is to always process the small jobs, resulting in this example in a loss of efficiency of factor 10.

5 RELATED WORK

Douglas Jensen proposed to associate to each job a value expressed as a function of time [21], resulting in time/utility functions to precisely define the semantics of soft real-time systems. The utility functions used in this work are similar to other proposals [13] and focus on simplicity for performance reasons.

In this paper, we compared the performances of on-line scheduling algorithms, using optimal clairvoyant algorithm as baseline. This methodology has been used in several previous research works. [7] proves that no on-line scheduling algorithm can have a competitive ratio greater than 1/4 when the loading factor is not bounded. The authors proved that 1/2 is a tight upper bound for jobs with zero laxity in 2-processors systems. [24] describes D^{over}, an on-line scheduling algorithm for overloaded uniprocessor systems. D^{over} provides the best achievable competitive ratio for non-clairvoyant algorithms. For proving upper and lower bounds on competitive ratios achievable by on-line algorithms, several techniques are discussed in [23]. Additionally, [23] proves that using competitive ratio as a performance measure is roughly equivalent to assume an omniscient adversary that has perfect knowledge of the algorithm and unlimited computing resources.

In order to identify on-line algorithms that have good real-life performance, a weakening of this adversary and/or a strengthening of the evaluated non clairvoyant algorithm is suggested. It was realized in [9, 17, 29] that randomization could (relatively) lower the power of the adversary since the decisions of the on-line algorithm are no longer certain. According to this hypothesis, randomized on-line algorithms can arguably be considered more efficient than deterministic ones, against different types of adversaries. In [22], it is shown that moderately increasing the speed of the processor used by a non-clairvoyant algorithm scheduler effectively gives this scheduler the power of clairvoyance. More interestingly, it is shown that there exist online scheduling algorithms with bounded competitive ratios on all inputs, and which are not closely correlated with processor speed. In [25], a strengthening of the on-line algorithm is considered. An on-line scheduling algorithm is said to be *speed-s optimal* if the algorithm can match the performance of an optimal clairvoyant algorithm with the same number of processors but having s times higher execution speed. Through the use of an analytic approach, it is demonstrated that EDF-ac[2] achieves speed-2 (resp. speed-3) optimality in overloaded uniprocessor (resp. multiprocessor) systems. In [6], the same speed factor metric is used. The scheduling algorithm *Earliest Deadline First with Virtual Deadlines* (*EDF-VD*) is proved to be *optimal* with respect to this metric. Compared to our study, their results are based on a *Mixed-Criticality implicit-deadline sporadic tasks*, i.e. firm real-time jobs that have specific inter-arrival date constraints and specific deadline values. A comparative study among algorithms that use different priority assignments is presented in [11]. The analysis is based on jobs values and jobs deadlines, but also different guarantee mechanisms, to improve the performance of a real-time system during overload conditions. The algorithms have been compared using the HVR metric, in two different jobs set where jobs characteristics were different in order to see how sensitive an algorithm is with respect to the jobs parameters. Authors observed that Highest Density First (HDF) algorithm, based on value density and corresponding to DVD1 algorithm in our paper, is the most effective in overloaded conditions, exhibits a very graceful degradation during overloads, and is not much sensitive to jobs set parameters. In [2], another comparative evaluation between on-line scheduling algorithms under overloaded conditions is done. In this study, jobs are soft real-time and a unique jobs set with arbitrary jobs characteristics was used for performance evaluation. It is concluded that DVD2 performs better and, on this basis, that DTD2 is an effective scheme and it is more suitable to operate under all operating loads than SDVD and/or EDF.

[2]EDF algorithm supplemented with a simple form of admission control.

CONCLUSION AND PERSPECTIVES

Intelligent Transportation Systems (ITS) require a careful optimization of the V2C data flow, taking into account temporal and value constraints. We have shown that this optimization obstacle is similar to a soft real-time scheduling problem. For a given ITS scenario, one has to decide on a particular scheduling algorithm, among the many existing scheduling algorithms, calling for a comparative evaluation that is representative of real-use cases. Unfortunately, real-use cases do not exist for now.

We proposed a methodology to generate workloads with increased diversity, which implies a better scope of evaluation. Our extensive evaluation experiments cover overloaded conditions with very high load rates (up to 1600%) that are expected to be encountered in realistic cases. We show that, among the set of evaluated algorithms, algorithm DTD1 is both robust and very efficient, outperforming other (non-clairvoyant) algorithms for every load rate. In addition, the performance bounds of an optimal clairvoyant algorithm have been estimated with a 2% relative error, providing some insights on the relevance of trying to overcome the performance achieved by DTD1.

The current outcome of this work is to provide a suitable mechanism to dynamically determine message values and time constraints. The keying of messages values will allow adaptation of data flow and must satisfy different, independent, and dynamic needs of services running in the Cloud. Some proposals, e.g., described in [10] and [27], are particularly interesting and require more thorough investigation. On the experimental side, our results will be validated on real-life platforms and simulators that are being developed in the automotive industry.

ACKNOWLEDGMENTS

This work is supported by Continental CDSF Grant Nr 180184, which is part of the eHorizon project.

REFERENCES

[1] Adwan Alanazi and Khaled Elleithy. 2015. Real-time QoS routing protocols in wireless multimedia sensor networks: Study and analysis. *Sensors* 15, 9 (2015), 22209–22233.

[2] Saud Ahmed Aldarmi and Alan Burns. 1999. Dynamic value-density for scheduling real-time systems. In *Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on*. IEEE, 270–277.

[3] MA Ameen, Ahsanun Nessa, and Kyung Sup Kwak. 2008. QoS issues with focus on wireless body area networks. In *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, Vol. 1. IEEE, 801–807.

[4] Hal R Arkes and Catherine Blumer. 1985. The psychology of sunk cost. *Organizational behavior and human decision processes* 35, 1 (1985), 124–140.

[5] Hagit Attiya, Leah Epstein, Hadas Shachnai, and Tami Tamir. 2010. Transactional contention management as a non-clairvoyant scheduling problem. *Algorithmica* 57, 1 (2010), 44–61.

[6] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo DAngelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. 2012. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *2012 24th Euromicro Conference on Real-Time Systems*. IEEE, 145–154.

[7] Sanjoy Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis Rosier, Dennis Shasha, and Fuxing Wang. 1992. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems* 4, 2 (1992), 125–144.

[8] Sanjoy Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis Rosier, and Dennis Shasha. 1991. On-line scheduling in the presence of overload. (1991).

[9] Allan Borodin, Nathan Linial, and Michael E Saks. 1992. An optimal on-line algorithm for metrical task system. *Journal of the ACM (JACM)* 39, 4 (1992), 745–763.

[10] Alan Burns, Divya Prasad, Andrea Bondavalli, Felicita Di Giandomenico, Krithi Ramamritham, John Stankovic, and Lorenzo Strigini. 2000. The meaning and role of value in scheduling flexible real-time systems1. *Journal of systems architecture* 46, 4 (2000), 305–325.

[11] Giorgio Buttazzo, Marco Spuri, and Fabrizio Sensini. 1995. Value vs. deadline scheduling in overload conditions. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*. IEEE, 90–99.

[12] Giorgio C Buttazzo. 2011. *Hard real-time computing systems: predictable scheduling algorithms and applications*. Vol. 24. Springer Science & Business Media.

[13] Giorgio C Buttazzo. 2011. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications (Real-Time Systems Series). (2011).

[14] Alin-Mihai Cailean, Barthélemy Cagneau, Luc Chassagne, Valentin Popa, and Mihai Dimian. 2014. A survey on the usage of DSRC and VLC in communication-based vehicle safety applications. In *Communications and Vehicular Technology in the Benelux (SCVT), 2014 IEEE 21st Symposium on*. IEEE, 69–74.

[15] Dazhi Chen and Pramod K Varshney. 2004. QoS Support in Wireless Sensor Networks: A Survey.. In *International conference on wireless networks*, Vol. 233. 1–7.

[16] Jian-Jia Chen. 2016. Scheduling Aperiodic Jobs on Uniprocessor Systems. [Powerpoint slides]. Retrieved from https://ls12-www.cs.tu-dortmund.de/daes/media/documents/teaching/courses/rts/aperiodic.pdf.

[17] Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. 1991. Competitive paging algorithms. *Journal of Algorithms* 12, 4 (1991), 685–699.

[18] Qingqing Gan and Torsten Suel. 2009. Improved techniques for result caching in web search engines. In *Proceedings of the 18th international conference on World wide web*. ACM, 431–440.

[19] Michael R Garey and David S Johnson. 1979. Computers and intractability: A guide to the theory of npcompleteness (series of books in the mathematical sciences), ed. *Computers and Intractability* 340 (1979).

[20] Cisco Visual Networking Index. 2019. Global mobile data traffic forecast update, 2017–2022 white paper. *White Paper* (Feb. 2019). https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.pdf

[21] E Douglas Jensen, C Douglas Locke, and Hideyuki Tokuda. 1985. A Time-Driven Scheduling Model for Real-Time Operating Systems.. In *RTSS*, Vol. 85. 112–122.

[22] Bala Kalyanasundaram and Kirk Pruhs. 2000. Speed is as powerful as clairvoyance. *Journal of the ACM (JACM)* 47, 4 (2000), 617–643.

[23] Richard M Karp. 1992. On-line algorithms versus off-line algorithms: How much is it worth to know the future?. In *IFIP Congress (1)*, Vol. 12. 416–429.

[24] Gilad Koren and Dennis Shasha. 1995. Dˆover: An Optimal On-Line Scheduling Algorithm for Overloaded Uniprocessor Real-Time Systems. *SIAM J. Comput.* 24, 2 (1995), 318–339.

[25] Tak-Wah Lam, Tsuen-Wan Johnny Ngan, and Kar-Keung To. 2004. Performance guarantee for EDF under overload. *Journal of Algorithms* 52, 2 (2004), 193–206.

[26] C Douglass Locke. 1986. Best-effort decision making for real-time scheduling. *(Ph. D Thesis), Computer Science Department, CMU* (1986).

[27] Tie Luo and Chen-Khong Tham. 2012. Fairness and social welfare in incentivizing participatory sensing. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*. IEEE, 425–433.

[28] Panos Papadimitratos, Arnaud De La Fortelle, Knut Evenssen, Roberto Brignolo, and Stefano Cosenza. 2009. Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation. *IEEE communications magazine* 47, 11 (2009).

[29] Prabhakar Raghavan and Marc Snir. 1989. Memory versus randomization in on-line algorithms. In *International Colloquium on Automata, Languages, and Programming*. Springer, 687–703.

[30] Daniel D Sleator and Robert E Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 2 (1985), 202–208.

[31] John A Stankovic, Marco Spuri, Marco Di Natale, and Giorgio C Buttazzo. 1995. Implications of classical scheduling results for real-time systems. *Computer* 28, 6 (1995), 16–25.

[32] Mohamed Younis, Kemal Akkaya, and Moustafa Youssef. 2010. Handling QoS Traffic in Wireless Sensor Networks. In *Encyclopedia on Ad Hoc and Ubiquitous Computing: Theory and Design of Wireless Ad Hoc, Sensor, and Mesh Networks*. World Scientific, 257–279.

A VALIDATION OF OBSERVED PERFORMANCE DIFFERENCE BETWEEN DVD1 AND DVD2 USING BOOTSTRAP METHOD

We want to assert that the observed performance difference between algorithms DVD1 and DVD2 is statistically significant. As shown in Figure 7, clearly none of HVR values obtained by all algorithms follows the distribution of a normal law. Consequently, ANOVA or t-student methods are not directly applicable.

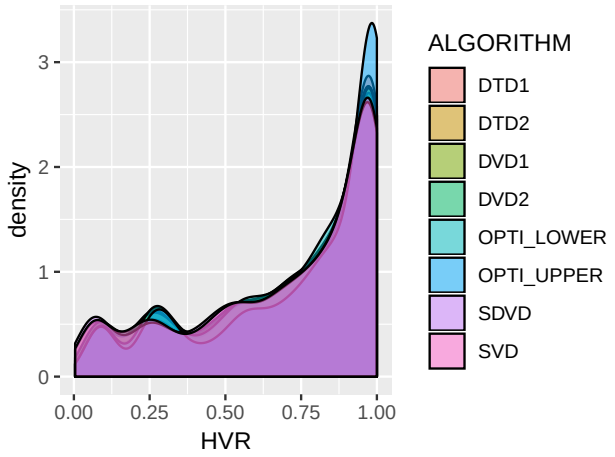


Figure 7: Distribution of HVR values of all algorithms for all (3000) simulated scenarios.

As an alternative, one may calculate the performance difference between algorithms two-by-two, and make an analysis of this difference. Unfortunately, the HVR difference between algorithms DVD1 and DVD2, plotted in Figure 8, doesn't follow a normal law distribution too.

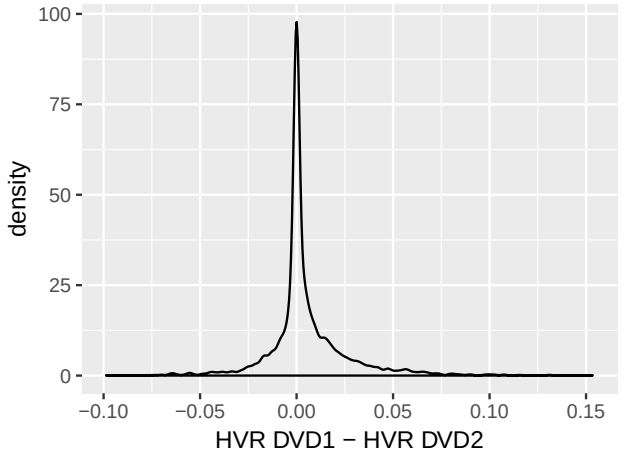


Figure 8: Distribution of paired difference HVR DVD1 - HVR DVD2 for all (3000) simulated scenarios.

Eventually, we decided to apply a bootstrap method. Let's denote by X, Y two column vectors such that X_i, Y_i is the HVR obtained respectively by the algorithm DVD1 and the algorithm DVD2 under the same scenario i. Because DVD1 algorithm outperforms DVD2 algorithm invariably of the load rate, we consider the results obtained over all (3000) generated scenarios. We calculate the performance difference of algorithm DVD1 vs algorithm DVD2 $D = X - Y$.

Now let's consider the following hypothesis *H0*: the algorithms DVD1 and DVD2 have no statistically significant performance difference. If *H0* holds, we should not be able to distinguish the results obtained from one or another algorithm.

We calculate the average difference obtained when randomly exchanging the algorithms' HVR values: $\sigma_i = D \circ R$, where R is a random vector who's coefficients are randomly sampled in $\{-1, 1\}$ and $(\circ)$ is the *Hadamard* product operator (coefficient-wise product, also known as *Schur* product or *entrywise* product). This process is repeated for $i \in 1, 2, .., N, N = 100000$.

The resulting distribution of obtained values $\{\sigma_1, \sigma_2, ..., \sigma_N\}$, shown in Figure 9, follows a normal distribution. It is a very good approximation of the distribution that one could expect when trying to reproduce the obtained results if hypothesis *H0* holds: the 95% confidence interval is in $I = \left[-2.57 \times 10^{-6}; 1.60 \times 10^{-6}\right]$. In our results, the mean performance difference is $mean(X1 - X2) = 5.0 \times 10^{-3}$ and this value doesn't belong to I, hence the hypothesis *H0* does not hold and we can conclude that our results are statistically significants.

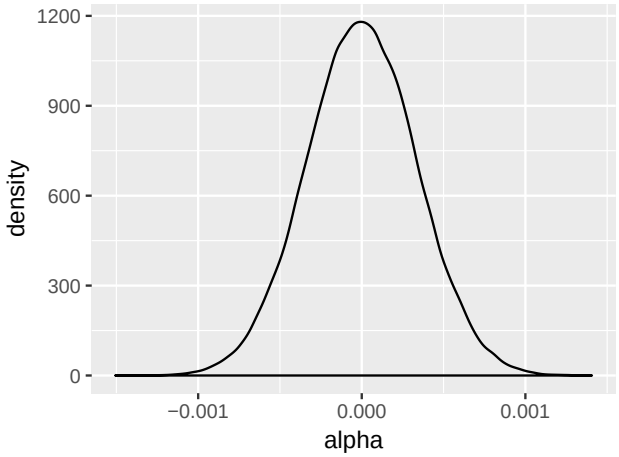


Figure 9: Distribution of the sampled values $\{\alpha_1, \alpha_2, ..., \alpha_N\}$ with $N = 100000$.