Resource Assignment in Vehicular Clouds

Mahmudun Nabi*, Robert Benkoczi* Sherin Abdelhamid†, and Hossam S. Hassanein‡
*Optimization Research Group,
University of Lethbridge, Alberta, Canada
m.nabi@uleth.ca, robert.benkoczi@uleth.ca
†Faculty of Computer and Information Science,
Ain Shams University, Cairo, Egypt
shereen@cis.asu.edu.eg
‡Queen‘s Telecommunication Research Lab
Queen’s University, Kingston, Ontario, Canada
hossam@cs.queensu.ca

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

I. Introduction

Cisco introduced the concept of *fog computing* to deal with Big Data analytics and applications in the Internet of Things [1], [2]. Fog computing attempts to improve the performance of systems through computations that are closer to the customers requesting the service rather than being located in far off data centres [3], [4]. The vehicular cloud (VC) is a particular implementation of fog computing. A vehicular cloud can support computations that are typically sent to a cloud system by applications running on mobile devices. Apple’s voice recognition system, Siri, is one such application.

Olariu et al. [5] introduced the concept of vehicular clouds about the same time when Cisco was promoting the notion of fog computing. In vehicular clouds, smart vehicles are equipped with components that have sensing, computing, and wireless communication capabilities which can be harvested for data storage, computing, infotainment, and sensing services [6], [7], [8]. Wireless technologies such as WAVE (wireless access for vehicular environment) which is based on the IEEE 802.11p standard, can be used by the vehicles to support Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communications. However, the vehicles are not owned by the service providers and they are a transient computation resource. Coordinating such computing resources requires solving scheduling problems with specific constraints. In vehicular clouds, incentives are offered to the vehicle owners to encourage renting their resources to the service providers, thus minimizing renting cost is one of the goals of the scheduler.

In this paper, we focus on the task scheduling problem in vehicular clouds. There is an impressive amount of literature on scheduling tasks in domains such as machine scheduling [9], distributed, and grid computing [10]. Most popular objectives for these task scheduling problems are focused on quality of service parameters such as makespan or lateness. In the vast majority of the problems, the computation resources are always available. This assumption has been relaxed in machine scheduling problems with availabilities [11] in which the processors are available to execute jobs only during certain moments in time. The objectives most studied in problems of scheduling with availabilities are minimizing maximum and total job completion time and minimizing maximum lateness. Scheduling problem with availabilities where preemption is not allowed makes the problem instances difficult to solve. In contrast, tasks scheduled on a particular resource in a vehicular cloud can be paused, transferred to another resource, and resumed on that resource. This is needed for example, when a particular vehicle leaves the cloud and all of the jobs assigned to it need to be transferred to other vehicles. In addition, for VCs with no infrastructure support to store the state of running processes, a task cannot be paused and resumed at a later time because the resource storing its state may leave the cloud. We are thus interested in a task scheduling problem with availabilities subject to a relaxed notion of non-preemption: once a task is started on a resource, it must be executed to completion without interruption, except when execution is transferred to another resource and is resumed immediately. We call this constraint *VC-preemption*.

We adapt the three field problem notation that is standard in the machine scheduling literature to our context: $T/M/C$, where T is an integer representing the number of tasks to be scheduled, M represents the machine environment which is I for identical machines or U for unrelated machines, and C is the constraint which is VC for VC-preemption with task deadlines. The problem objective is to minimize the cost of the schedule.

978-1-4673-8999-0/17/$31.00 ©2017 IEEE

To the best of our knowledge, no previous study of a machine scheduling problem with such a constraint has been carried in the literature we surveyed. Our contributions are summarized as follows:

- We show that the problem of scheduling a single task on a set of unrelated machines to minimize scheduling cost and subject to a task deadline and VC-preemption ($1/U/VC$) is NP-hard, by exploiting the connections of the problem with knapsack cover (KC) [12].
- We provide a fully polynomial time approximation scheme (FPTAS) for problem $1/U/VC$ by extending the idea used in an FPTAS for knapsack cover [13] to our problem with VC-preemption.
- We describe a natural greedy algorithm for problem $1/U/VC$. We note that our greedy algorithm is trivially optimal for the version with identical machines, $1/I/VC$.
- We provide a standard greedy algorithm to schedule n tasks on unrelated machines with VC-preemption (problem $n/U/VC$) that repeatedly calls a procedure to solve $1/U/VC$.
- We give a simple and powerful lower bound on the cost of the optimal solution for scheduling n tasks by solving a knapsack cover problem fractionally [14]. We use this bound to calculate approximation ratios of our algorithm for problem $n/U/VC$ on an extensive set of problem instances.
- We perform a comprehensive empirical evaluation of our task scheduling algorithm for problem $n/U/VC$ with both the greedy and PTAS procedures for solving $1/U/VC$, using both synthetically generated data and real data extracted from vehicle mobility traces and grid workload traces. We note that the scheduling problems studied here are off-line, the set of tasks and the availability of the resources are known at the start of the simulation. Our results indicate that the $n/U/VC$ algorithm with the greedy procedure for $1/U/VC$ and the proposed lower bound on the optimal solution are extremely powerful. On the synthetic problem instances, the average gap between the lower bound of the schedule cost and the solution returned was less than 2.5% and it was no larger than 25% on the real data instances. These findings are pivotal for the evaluation of a vehicular cloud task scheduler in the on-line setting where tasks and resource availabilities become known with time.
- We provide the first, as far as we know, quantitative evidence on the processing capability of a vehicular cloud using real life grid workload traces and considering the transient nature of the cloud resources and the specifics of task VC-preemption constraints. We observe that more than 92% on average of the grid processes were scheduled successfully on the vehicular cloud in the most constrained of the instances, and, when given some slack, a vehicular cloud can serve all but a handful of the most compute intensive jobs which need to be offloaded to the classical cloud.

The remainder of this paper is organized as follows. In Section II, the system model for task scheduling in VC is described. Complexity of the problem and proposed algorithms are discussed in Section III. In Section IV, experimental results are presented. Section V concludes the paper.

II. System Model

We consider a vehicular cloud consisting of a set of m vehicles available during a determined time interval. We denote vehicles by symbol i. Each vehicle i has a processing speed (processing capability) denoted $\alpha(i)$. It represents the number of operations the processor can handle per unit of time. The start and end of the availability interval of vehicle i is denoted t_i^- and t_i^+, respectively. A constant rental cost per unit of time, $C(i)$, is associated with vehicle i.

Each processing task k has a processing requirement ρ_k, measured in machine instructions, which needs to be served by the resources in the vehicular cloud (VC). We consider an offline environment where the problem instance is completely available. We assume that all the tasks have a common deadline T, representing the maximum allowable duration each task can take to finish. We note that this requirement can be made more general to include task specific deadlines. The algorithms proposed here can handle this extension.

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

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

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

III. Resource assignment in vehicular clouds

The single task allocation on a set of vehicles sub-problem can be related to the knapsack cover (KC).

Problem 1: Knapsack Cover (KC): Given is a set of n items $X = \{1, ..., n\}$ with a cost $c_i \in \mathbb{Z}^+$ and a size $s_i \in \mathbb{Z}^+$ for each item and a demand $D \in \mathbb{Z}^+$. The goal is to find a minimum cost subset of items $F \subseteq X$ subject to the constraint that the total size of F is at least as large as D.

Given an instance of KC, we construct an instance of $1/U/VC$ as follows. Each object in KC corresponds to a machine with an availability interval and speed calculated so that it equals the size of the object. The availability intervals are chosen so that they are consecutive in some arbitrary order of the objects. The single task has the processing requirement equal to D. We double the number of machines by adding exactly one machine with the availability interval equal to that of a machine corresponding to a KC object but with zero speed and zero cost. It can be argued that any KC solution corresponds to a feasible schedule for $1/U/VC$ consisting of machines corresponding to the objects in the KC solution plus possible other machines with zero cost and speed that are chosen to satisfy the VC-preemption constraints.

A. The algorithms

We partition the time into consecutive time intervals. Each time interval $[t_1, t_2]$ is such that t_1 and t_2 represent the starting or the ending point of the availability interval for some machine and no other availability interval starts or ends between t_1 and t_2. In fact, the set of available machines does not change between t_1 and t_2. We call these intervals **partitions**. The available processors in each partition are called the **objects** and are denoted by o_{ij}.

Definition 1: An arbitrary partition P_i is a set of objects $\{o_{i1}, o_{i2}, ...\}$ such that each object $o_{ij} \in P_i$ has a length of $\delta_i = Pt_i^+ - Pt_i^-$, where δ_i represents the partition length, Pt_i^- is the start time, and Pt_i^+ is the end time of partition P_i.

We denote the set of partitions and set of objects as $\mathcal{P}$ and $\mathcal{O}$, respectively. The end time of the last partition corresponds to the deadline T. The resource capacity (s_{ij}) and the total access cost (c_{ij}) of an object $o_{ij} \in P_i$ is computed by multiplying its processing speed per unit of time and the cost per unit of time, respectively, with its length. Moreover, we introduce the notion of scheduling intervals, collectively represented by set $\mathcal{I}$, where each interval $\mathcal{I}_i$ is defined as follows:

Definition 2: A scheduling interval $\mathcal{I}_i$, denoted by $[Pt_s^-, Pt_t^+]$, is a collection of objects from partitions $\{P_s, P_{s+1}, ..., P_t\}$ where Pt_s^- and Pt_t^+ are the start time and the end time, respectively, of this interval. The scheduling interval length $|\mathcal{I}_i|$ is measured as the number of partitions from P_s to P_t.

A scheduling interval corresponds to the actual period of time that a task is scheduled on the vehicular cloud processors. If the scheduling interval for a task is known, then the actual minimum cost schedule for that task can be obtained by solving an instance of the multiple choice knapsack cover problem.

Problem 2: Multiple Choice Knapsack Cover Problem (MCKCP): Given n classes $N_1, N_2, ..., N_n$ of items, where each item $j \in N_i$ has a non-negative size, s_{ij} and a non-negative cost c_{ij}, and given a demand $D \in \mathbb{Z}^+$, the objective is to choose exactly one item from each class such that the total size of the chosen objects is at least D and the total cost of the objects is minimized.

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

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

B. PTAS for $1/U/VC$ problem

The procedure enumerates all $O(m^2)$ scheduling intervals. For a given scheduling interval, we need to solve an instance of MCKCP. Let the scheduling interval consist of partitions $P_1, \ldots, P_z$. A scaling based fully polynomial time approximation scheme (FPTAS) for minimum knapsack cover (KC) problem is proposed in a thesis by Islam et al. [13]. Their proposal uses a dynamic programming (DP) approach for solving the KC problem. We extend this scheme to the MCKCP problem.

We assume first that the cost of objects in the MCKCP are all integer. We denote the DP sub-problem by $S(i, c)$, which represents the maximum total size of objects from partitions $\{1, ..., i\}$ for $i \leq z$ given a budget c used to pay the costs of the objects. The DP sub-problem can be obtained recursively,

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

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

Let c_{max} be the highest cost of an object among the z partitions. Then a feasible solution cannot have cost larger than zc_{max}. If there are a total of n objects, then the total running time taken for building the DP table is $\mathcal{O}(nzc_{max})$. Once the DP table $S(,)$ is built, the optimal solution can be found by looking at the entry that satisfies the demand with minimum cost c. Thus, we can say that $1/U/VC$ can be solved in time that is pseudo-polynomial in the size of the problem.

The above pseudo polynomial time algorithm can be converted into a FPTAS using cost scaling. Suppose we can guess the value c^* representing the maximum cost of an object used by the optimal solution to the MCKCP (we can run this procedure n times, once for each possible value of c^*, and keep the best solution). The guessed value c^* can be used as a simple lower bound on the cost of the optimal solution in the proof of the $(1+\varepsilon)$ approximation factor of our algorithm. We choose a scaling factor $k^* = \frac{\varepsilon c^*}{z}$ and we scale every object j from partition i with cost $c_{ij} \leq c^*$ to integer cost $c'_{ij} = \lfloor \frac{c_{ij}}{k^*} \rfloor$. It can be shown that the dynamic programming algorithm from (1) is an FPTAS when executed over the scaled costs c'_{ij}. We can state the following theorem. Details are omitted due to space constraints.

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

C. Greedy algorithm for $1/U/VC$ problem

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

IV. Experimental Analysis

We evaluated our scheduling algorithms in three different scenarios: i) fixed number of vehicles and varying number of tasks, ii) varying the ratio between tasks with a large demand (long tasks) and tasks with a small demand (short tasks) while maintaining the ratio between total demand and the total available processing resource constant, and iii) task profiles extracted from real grid workload traces.

We generated the test instances for the first two scenarios randomly. We used a normal distribution for the duration of the availability intervals for the machines. To choose the starting time for the availability intervals, we chose a particular moment within the simulation time window and we generated the starting point of the machine availabilities using the normal distribution around it. We used a uniform distribution between values 1 and 10 for the speed and cost per unit of time for each vehicle. We used a bi-modal distribution for task requirements to create sets of short and long tasks. We call a task short (long) if its processing requirement needs a fifth of (three times) the average duration of a partition (see Section III-A) to serve at the average machine speed.

a) Scenario i: In these experiments, we vary the number of tasks while keeping the resources fixed. We categorized the test instances into two groups: small-size and large-size, based on the values of m (the number of machines) and n (the number of tasks). For small-size instances m and n values are taken from set $\{5, 8, 10\}$ and $\{5, 10, 20, 30, 50, 100, 200\}$, respectively. For large-size instances $m \in \{20, 30, 50\}$ and $n \in \{10, 20, 30, 50, 100, 200\}$. We generate 10 random machine instances for each value of m and 10 random sets of tasks for each value of n. We solved instances corresponding to all possible combinations of values for m and n and we report the average over all of the instances with the same value of n.

We propose the following simple lower bound on the cost of the optimal task schedule. Consider all machines $i \in \{1, \ldots, n\}$ indexed in non-decreasing order of their cost per speed ratio ($\frac{C(i)}{\alpha(i)} \leq \frac{C(i+)}{\alpha(i+)}$ for all $1 \leq i \leq n-1$). This ratio represent the cost for serving a unit of demand. Given a problem instance I, we calculate the cost of serving the total demand of I by the cheapest resources. Clearly, the optimal schedule cannot do better than this. If $C_A(I)$ represents the cost of the solution returned by algorithm A on instance I and $C'(I)$ is the value of this lower bound, then the approximation ratio of algorithm A on instance I is $\frac{C_A(I)}{C'(I)} \geq 1$.

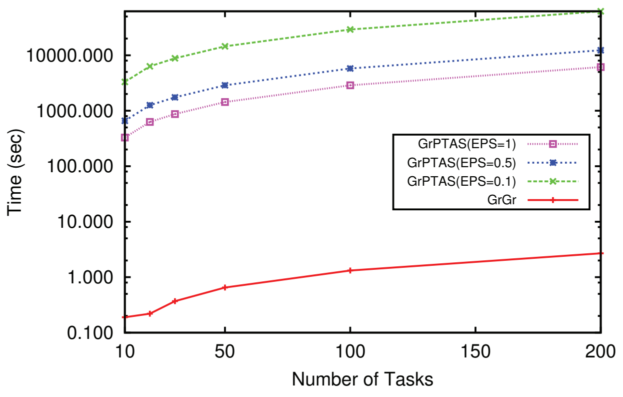


Fig. 2. Time comparison for large test instances.

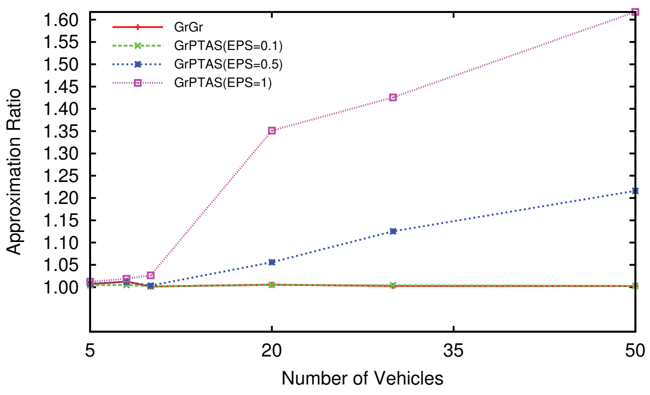
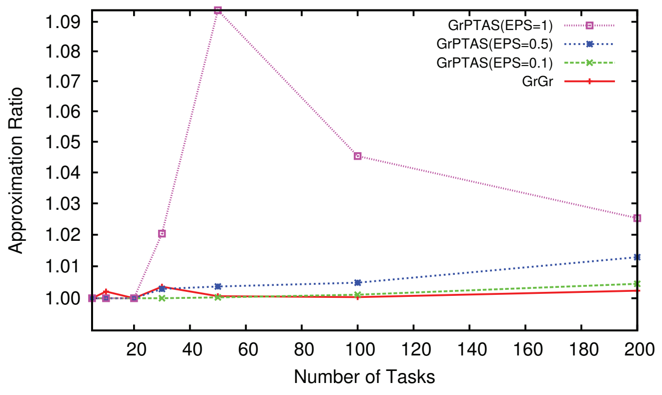


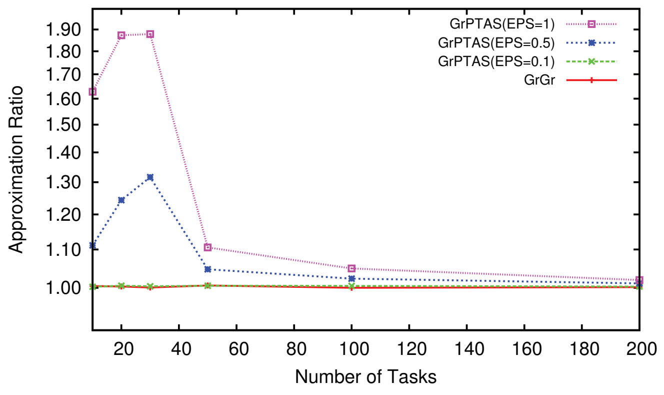
Fig. 3. Approximation ratio w.r.t. number of vehicles.

Fig. 1 shows the approximation ratio as a function of the number of tasks. Each line in this figure represents the algorithm used for solving the $1/U/VC$ instance. Fig. 1a and 1b depict the approximation ratio of small and large instances, respectively. Additionally, Fig. 2 shows the running times (in seconds) for large instances where *GrGr* is significantly faster than *GrPTAS* for all ε values. We ignored the time comparison for small instances as they are almost identical for all the algorithms. Moreover, Fig. 3 shows the performance of the algorithms when the number of vehicles increases. As expected, the cost of the schedule returned by *GrPTAS* for ε = 0.1 outperforms *GrGr* in an expense of higher running time.

b) Scenario ii: In this scenario we vary the ratio between the total processing demand of the tasks and the total processing capability of the machines. We call this ratio the *constraint level* and we denote it by γ. We used five different constraint levels: 10%, 25%, 50%, 75% and 100%. We generated machine instances for $m \in \{8, 10, 15, 20\}$ and task instances for five different ratios β between the total demand originating from long tasks and the total demand originating from short tasks for $\beta \in \{9, 1, 2, 1/2, 1/9\}$. The

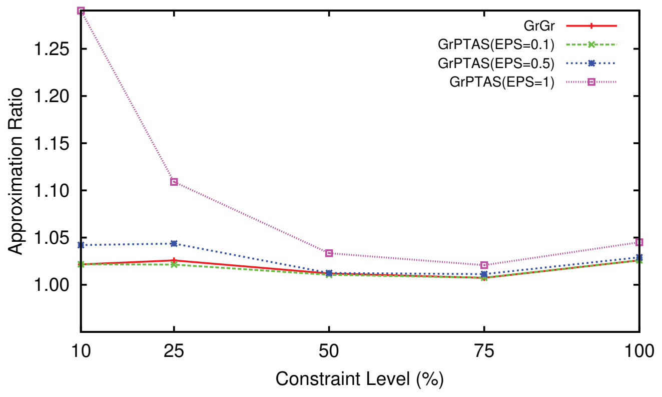


(a) Approximation ratio of small test instances.

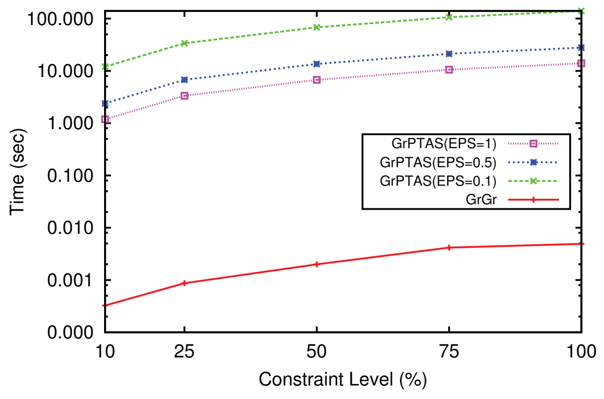


(b) Approximation ratio of large test instances.

Fig. 1. Performance of greedy algorithms on different size test instances



(a) Approximation ratio for different constraint levels



(b) Average time for different constraint levels

Fig. 4. Performance of greedy algorithm for different constraint levels

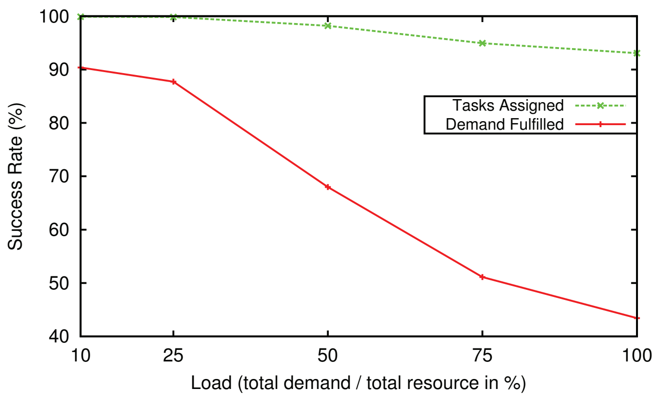
task instances were generated so that the constraint level remains constant. For each (γ, β) pair, 10 random instances were generated, for a total of 250 problem instances.

Figures 4a and 4b depict the performance ratio and the average running time for different load conditions. We see that the performance of *GrGr* and *GrPTAS* for $\varepsilon = 0.1$ is almost identical in terms of cost minimization. However, *GrGr* is significantly faster than all ε values of GrPTAS as the load factor increases.

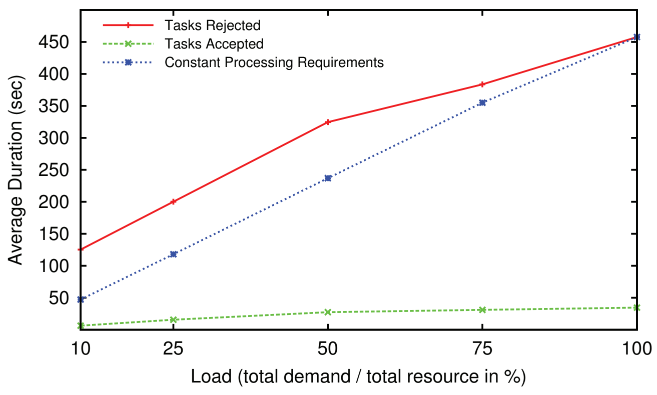
c) Scenario iii: In this experiment we have used a database of mobility traces collected from taxis in San Francisco. It contains GPS coordinates of approximately 500 taxis collected over over 30 days [15]. We assumed that taxis (vehicles) from cabspotting database are equipped with processors. These processors can be organized to form a vehicular cloud around a fixed reference point and computational tasks can be offloaded to this VC for processing. To generate the machine instances, we selected a reference point in San Francisco and we computed the availability intervals according to the time interval in which taxis are within some radius of the reference point.

For this experiment, the simulation time window was chosen equal to one day (86400 seconds). We selected 10 slices of one day duration each from the taxi mobility database collected within 100m radius of the City Hall. We selected 10 sets of task requirements based on the running time of processes from 10 randomly chosen slices of one day duration each of the DAS-2 system workload trace available from the Grid Workloads Archive (GWA) [16]. The instances extracted are extremely large, and so we have executed only the *GrGr* scheduler as it outperformed the GrPTAS algorithm in previous experiments. For each machine availability / task requirement pair, we assigned processor speeds uniformly at random to obtain a particular constraint level ranging from 100% (heavily loaded) to an instance of 10% (lightly loaded).

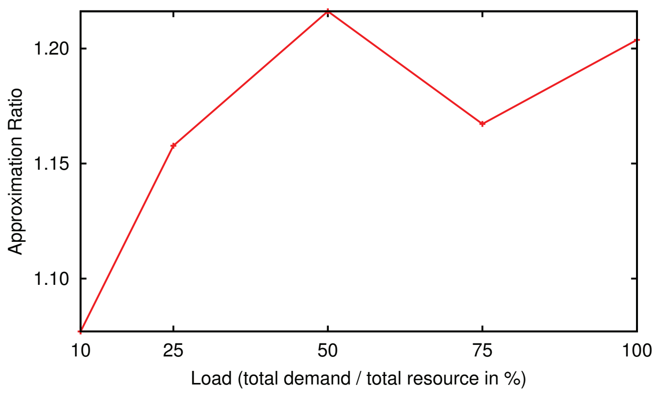
Fig. 5a depicts the success rate in terms of the fraction of demand fulfilled and the fraction of tasks completed as a function of the system load (load factor). Additionally, Fig. 5b shows the average duration of both tasks accepted and tasks rejected. Since we relax the constraint level in the graph from Fig. 5b by increasing the randomly assigned speed of the processors, we represent the line corresponding to the duration of a constant task requirement. We thus notice that the average duration of the accepted tasks increases with the decrease in constraint level up to the 50% mark. Comparing figures 5a



(a) Success rate



(b) Average duration



(c) Approximation ratio

Fig. 5. Performance of *GrGr* in real life scenario.

and 5b indicates that, *GrGr* was able to schedule most of the short tasks successfully whereas it failed to schedule some long tasks. Moreover, increasing the speed of the vehicles does not necessarily allow the vehicular cloud to process more tasks in some cases. Fig. 5c depicts the approximation ratio of the scheduler which is worse than in the case of randomly generated task requirements. We remark that, for any real life system, the tasks rejected by the vehicular cloud can be assigned to a fixed cloud infrastructure.

V. Conclusion

Vehicular cloud systems implement the concept of fog computing and will play a significant role in the Internet of Things. We argued that task scheduling in vehicular clouds, unlike in any other distributed computing environment, requires the solution to a machine scheduling problem with special constraints. We provide a first theoretical and empirical analysis of the scheduling problem and we present concrete measurements on the computational capacity of vehicular clouds. Our findings may not only prove useful to engineers considering an actual vehicular cloud deployment, but also for further research on algorithms for concrete resource scheduling problems in vehicular clouds.

Acknowledgments: Robert Benkoczi acknowledges the support received for this research from an NSERC Discovery Grant. The authors would like to thank the anonymous reviewers for their thoughtful comments which improved the quality of this paper.

References

[1] F. Bonomi, “Connected vehicles, the internet of things, and fog computing,” in *The Eighth ACM International Workshop on Vehicular Inter-Networking (VANET), Las Vegas, USA*, 2011, pp. 13–15.
[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.
[3] R. Deng, R. Lu, C. Lai, and T. H. Luan, “Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing,” in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 3909–3914.
[4] A. Destounis, G. S. Paschos, and I. Koutsopoulos, “Streaming big data meets backpressure in distributed network computation,” in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.
[5] S. Olariu, I. Khalil, and M. Abuelela, “Taking vanet to the clouds,” *International Journal of Pervasive Computing and Communications*, vol. 7, no. 1, pp. 7–21, 2011.
[6] S. Abdelhamid, H. S. Hassanein, and G. Takahara, “Vehicle as a resource (VaaR),” *IEEE Network Magazine*, vol. 29, no. 1, pp. 12–17, 2015.
[7] S. Olariu, M. Eltoweissy, and M. Younis, “Towards autonomous vehicular clouds,” *ICST Transactions on Mobile Communications and Applications*, vol. 11, no. 7-9, pp. 1–11, 2011.
[8] M. Whaiduzzaman, M. Sookhak, A. Gani, and R. Buyya, “A survey on vehicular cloud computing,” *Journal of Network and Computer Applications*, vol. 40, pp. 325–344, 2014.
[9] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys, “Sequencing and scheduling: Algorithms and complexity,” *Handbooks in operations research and management science*, vol. 4, pp. 445–522, 1993.
[10] F. Dong and S. G. Akl, “Scheduling algorithms for grid computing: State of the art and open problems,” Queen’s University, Tech. Rep., 2006.
[11] G. Schmidt, “Scheduling with limited machine availability,” *European Journal of Operational Research*, vol. 121, no. 1, pp. 1–15, 2000.
[12] M. R. Garey and D. S. Johnson, *Computers and intractability*. wh freeman New York, 2002, vol. 29.
[13] M. T. Islam *et al.*, “Approximation algorithms for minimum knapsack problem,” Ph.D. dissertation, Lethbridge, Alta.: University of Lethbridge, Dept. of Mathematics and Computer Science, c2009, 2009.
[14] G. V. Gens and E. V. Levner, “Computational complexity of approximation algorithms for combinatorial problems,” in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 1979, pp. 292–300.
[15] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, “CRAWDAD dataset epfl/mobility (v. 2009-02-24),” Downloaded from http://crawdad.org/epfl/mobility/20090224, Feb. 2009.
[16] Advanced School of Computing and Imaging (ASCI). (2005) GWA-T-1 DAS2. GWF format in Grid Workloads Archive. [Online]. Available: http://gea.ewi.tudelft.nl/datasets/gwa-t-1-das2