Automated Feedback-Based Vertical Elasticity for Heterogenous Soft Real-time Workloads

Yu-An Chen*, Andrew J. Rittenbach†, Geoffrey Phi C. Tran†*, John Paul Walters†, and Stephen P. Crago*†

*Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089
Email: {chen116, geoffret}@usc.edu

†Information Sciences Institute
University of Southern California, Arlington, VA 22203
Email: {aritten, gtran, jwalters, crago}@isi.edu

***Abstract*—Cloud computing provides a pool of highly available resources for applications to offload their tasks to, but new applications such as coordinated lane change assistance used in connected car system have strict timing requirements that cannot be met by offloading tasks only to the cloud. Fog computing reduces latency by bringing the computation from remote datacenter to local fog servers, which are connected in close proximity with the clients. Although fog computing lowers the latency for transferring data, load balancing among fog servers still needs to be addressed for better timing performance. The challenges include large number of tasks, mobility of the clients, and the heterogeneity of the fog servers. In this paper, using connected car systems as a motivating application, we first show that we can utilize mobility patterns of vehicles to perform periodic load balancing in fog servers. We then present a task model that solves the scheduling problem at the server level instead of device level. And finally we formulate a load balancing optimization problem for minimizing deadline misses and total runtime for connected car system in fog computing. We show that it outperforms some common heuristics such as weighted round robin, active monitoring, and throttled load balancer.**

***Keywords*-fog computing; load balancing; real-time; task distribution; deadline misses; connected car; mobility prediction**

I. Introduction

Although cloud computing provides a platform for applications to offload their tasks for its high availability and fast processing speed, the physical distance between the datacenter and the end users introduces latency that is too large for cloud computing to be effective for some time-sensitive applications.

Fog computing is a new paradigm that pushes the processing to the end users[1]. Unlike cloud computing which places servers in a centralized datacenter, fog computing distributes servers among the edge that is close to the end users. By offloading and running tasks on the fog servers, latency, as well as the traffic on the backbone network, is reduced.

Even though fog computing provides a solution to latency in relaying data, resource allocation and load balancing while considering deadline misses and total runtime are issues that still need to be addressed[2]. In this paper, we use connected car systems as a motivating application to explore load balancing problem in fog computing. Applications that run in connected car system can benefit from fog computing[3]. For example, coordinated lane change assistance[4][5] can utilize local fog servers to process data sent from nearby cars and respond quickly[3]. To perform load balancing in fog computing, we need to consider several characteristics of fog computing that are different from cloud computing. First is the heterogeneous nature of fog servers where each fog server can have different computation capability and connectivity[1]. And Since the idea of fog computing is to process data locally for faster response, the scale of fog system we are looking at is relatively small compared to a datacenter. The smaller scale of the fog system enables more precise approaches that would otherwise be inappropriate in a large scale cloud environment. Lastly, mobility of the clients is also a feature that should be addressed. The locations of the clients in cloud computing are relatively static compared to those in the case of moving vehicles in a connected car system. In fog computing, the relationship between servers and clients depends on their instantaneous locations. It can be beneficial to use the information about the location and mobility of the cars when trying to perform load balancing. For example, if we can predict the travel patterns of cars, we can pre-allocate resources to the best-fit server to achieve lower processing time and avoid deadline misses. In this paper, we first develop a mobility prediction algorithm and then propose a fog task model followed by a load balancing optimization problem formulation for connected car systems in fog computing.

The main contributions of the paper are:

1) We propose a task model that provides a platform to develop load balancing algorithm that does not require the knowledge of how an individual task is scheduled by moving the scheduling problem from device level to server level. It is impractical to obtain the scheduling information about each task because the

traffic of the network make the arriving order of tasks a stochastic process and the problem size increase more rapidly.

2) We propose an optimization problem formulation for load balancing that minimizes deadline misses and total runtime for connected car system in fog computing.

The rest of the paper is organized as follows: we review existing works in load balancing for mobile clients in fog computing in Section II. In Section III, we present a simple linear mobility prediction algorithm with high accuracy to show that online task scheduling is beneficial in fog computing. We present our task model and optimization problem formulation in Section IV. In Section V, we analyze the performance of the optimization problem formulation. Conclusions and future work are discussed in Section VI.

II. Related Work

In the work of Oueis et al. [6], they construct two optimization problems that minimize total transmitting power and total computation time respectively. In their work, tasks can be partitioned to servers and constrainted to hard deadlines, whereas tasks are indivisible and have soft deadlines in our work. Zeng et al. [7] formulates a linear program that aims to minimize the average task completion time, but we also focus on avoiding missing deadlines for each task in our formulation.

Hong et al. [8] propose a programming model that uses the workload to dynamically scale the fog system in order to provide enough resources. Their example applications include vehicle-to-vehicle video streaming and traffic monitoring. The difference between our work and theirs is our work consider deadline constraint and placements of individual task to the fog servers while theirs does not.

Takayuki et al. [9] use routing for smart cars as the application that is being offloaded to the fog servers. Their load balancing optimization formulation focuses on minimizing aggregated task finishing time under an energy constraint. The difference between our work and theirs is that they do not impose capacity or resource constraint on the servers while we do.

Hong et al. [10] use the car's mobility patterns to predict its future location and then forward the tasks to the fog server that is associated with predicted location so the processing can start without any delay when the car arrives. The tasks include gathering and processing the data from local sensors. Their work has similar approach to our work in that they utilize mobility but their system focuses on only single client and does not impose capacity/resource constraints on the servers while we work with a multi-clients system that have various contraints on the servers.

In the work of Wang et al. [11], they use face recognition running on mobile devices as the motivating application. Their approach is to treat tasks coming from mobile devices as a tree graph, and then solve for the optimal matching of those task nodes to the physical nodes. Their solution is to assign weighted costs for placing an application node to a physical fog node then find the optimal mapping that minimizes total cost. The difference between our work and theirs is our work accounts for timing constraints and multi-user models while theirs does not.

Li et al. [12] solve the problem of how to partition tasks for local servers and remote cloud to process and then allocate resource based on the partitioning. Video streaming on mobile devices is the example application. They propose an optimization formulation to minimize finishing time, cost of processing, and bandwidth usage. One major difference between our work and their work is that their formulation does not allow any deadline misses while our work tries to minimize deadline misses.

III. Mobility Pattern Prediction

A. Connected Car System Model in Fog Computing

Many works have been done in mobility prediction in vehicles using methods such as Markov Chain[13] or probability distributions[14]. In this section we will show that even a simple linear model can give high prediction accuracy such that task allocations based on mobility prediction is practical.

First we present a model for a connected car system in fog computing. An example model is illustrated in Figure 1. In Figure 1, A, B, and C represent three different fog servers and each car is counted as a client. Each fog server manages the cars that are within its wireless range. We assume the wireless communication protocol used between cars and servers is 802.11p, Wireless Access in Vehicular Environments(WAVE)[15], which has communication range of at least 500 meters or above[16]. We also use a hexagon to indicate the wireless coverage area of a fog server instead of omnidirectional coverage because hexagonal cell shape approximation is more suitable for implementing wireless network system[17].

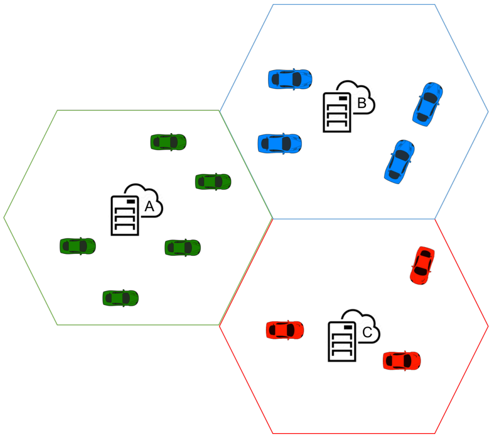


Figure 1: Example Fog Computing Model

B. Linear Mobility Pattern Prediction

We propose a linear mobility prediction algorithm that uses just previous location and current location of a car to predict its future location. Assuming each car updates its location every θ seconds, when a car updates its location at time t, the corresponding fog server uses the car's previous location and timestamp to calculate the car's current speed and direction. The predicted location for the car will be the position of the car at time $t + \theta$ with current speed and direction. If the predicted location is outside of the current fog server's wireless range, then the algorithm will determine which server the car will travel to by finding the neighboring server that is the closest to the cars predicted location. Figure 2 shows a graphical illustration of the prediction algorithm.

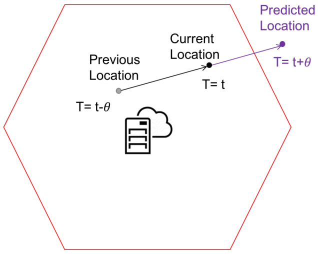


Figure 2: Mobility Prediction Algorithm

We test our linear prediction algorithm with real GPS data of taxis gathered in Rome, Italy[18]. The update period for the dataset is about 7 seconds. We divide up the region into a system with 7 fog nodes each with radius of 500 meters, as shown in Figure 3. We use our linear prediction algorithm to keep track of the taxis' mobility pattern, and notify the system when the algorithm anticipates a taxi is leaving or entering the wireless coverage of the center fog node by predicting which of the 6 regions the taxi is heading toward or coming from. The result is shown in Figure 4, where the blue lines indicate correct predictions and red lines indicate incorrect predictions. We are able to obtain a 94.89% accuracy as shown in Table I. In the next section, we will present a task model based on the model presented in this section, and then develop an optimization problem formulation for load balancing that minimizes deadline misses and total runtime.

Table I: Linear Prediction Numerical Result

Correct Preditction Count	2505
Total Prediction Count	2640
Accuracy	94.89%

IV. Load Balancing in Fog computing

A. Fog Computing Task Model

In section III, we show that vehicles' mobility pattern can be predicted with high accuracy. For a application such as lane change assistance in connected car, we want to have the data ready for a car when it connects with a new fog server. With the ability to predict which server a car is traveling to, we can perform load balancing and process the tasks before the cars arrive at their predicted locations.

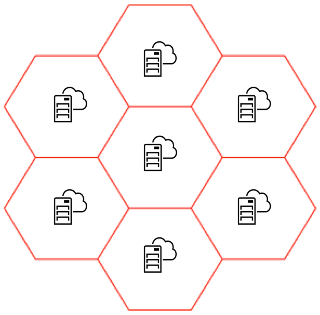


Figure 3: A Fog System with 7 Fog Nodes

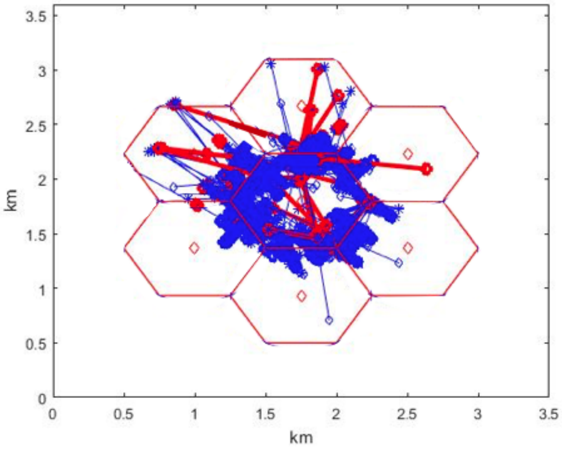


Figure 4: Linear Prediction Graphical Result

The task model for connected car system in fog computing is composed of cars, local fog servers and a remote cloud server. The local servers are inter-connected and also connected to the cloud server. A simple system can be illustrated using Figure 5. In Figure 5, there is one cloud server and three local fog servers, A, B and C. Each fog server is connected to the cars that sit in its wireless range, indicated by hexagons with different colors. We treat the workload from each car as one task. Those tasks are passed through the nearest servers, which can either keep the tasks and do the processing or distribute them to other servers that have extra capacities. So in Figure 5, 5 tasks are initially passed to server A, 3 tasks are initially passed to server B and, 4 tasks are initially passed to server C, and the cloud server does not get any initial task.

The general graphical representation of the task model is shown in Figure 6. The local servers are interconnected, so they can distribute their tasks among themselves or the remote cloud. In other words, all servers can both accept tasks from other servers and offload tasks to other servers. Each server can have different resources and constraints. Table II shows the variables used in the task model. In Table II, k is the number of servers in the system. N_i is the number of tasks initially passed to server i. Figure 5 shows an example system with $k = 4$, $N_A = 5$, $N_B = 4$, $N_C = 3$, and $N_{cloud} = 0$. B_{ij} is the link rate between server i and server j. C_j is the maximum number of tasks server j can handle. x is the number of CPU cycles required to process each task assuming all tasks consume equal amount of bandwidth. f_j is the CPU frequency of server j. D is the deadline for each task. τ is the allowed processing time for each server. The reason τ is needed is because we are doing load balancing based on mobility prediction which is calculated periodically. Every time the mobility prediction is updated, load balancing should be performed. In other words, we are restricting the time to process tasks to be less than the deadline so the remaining time can be used for running mobility prediction and load balancing. So $D - \tau$ can be seen as the overhead budget reserved for executing mobility prediction and load balancing. Finally, n_{ij} is the optimizing variable to be solved that tells how many tasks should be offloaded from server i to server j.

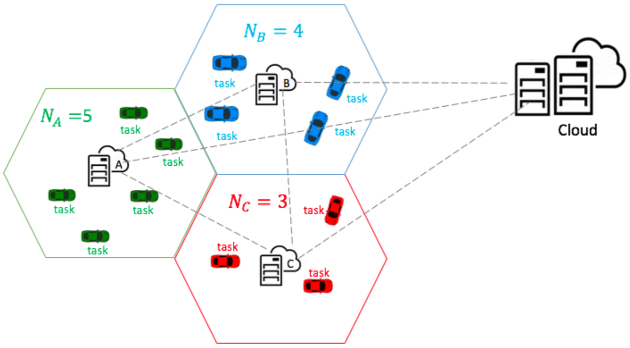


Figure 5: Task Model Example

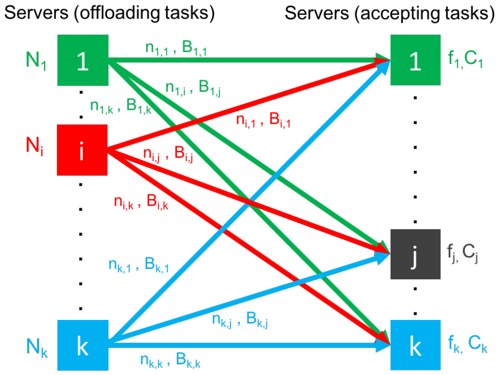
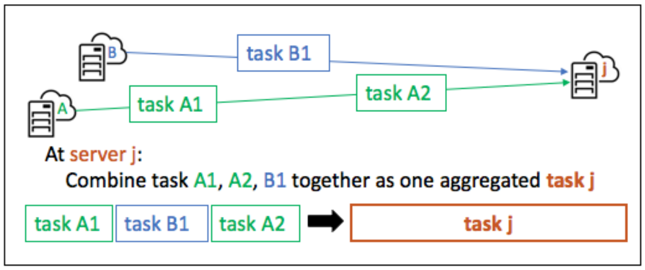


Figure 6: Task Model in Fog Computing

Since we do not know how a server schedules the tasks coming from other servers and the arrival order of those incoming tasks, it is impossible to calculate the completion time for each task that is offloaded to a server. So instead of trying to perform load balancing at the task level, we propose to execute load balancing at the server level by treating the tasks offloaded by other servers as one large aggregated task as illustrated in Figure 7. By doing so, we are able to calculate the completion time of the large aggregated task at each server disregarding how that server schedule its tasks and the arrival order of the incoming tasks.

Table II: Parameters & Variables in the Task Model

Parameters for Servers and Tasks	
k	Number of servers
N_i	Number of initial tasks for server i
B_{ij}	Link rate between server i and server j (tasks/sec)
C_j	Capacity of server j (tasks)
x	Number of CPU cycles required for each task (Mcycles)
f_j	CPU frequency of server j (MHz)
D	Deadline for all tasks (sec)
τ	Allowed processing time for each server (sec)
Optimizing Variables	
n_{ij}	Number of tasks distributed from server i to server j



At server j:
Combine task A1, A2, B1 together as one aggregated task j

Figure 7: Aggregated Task Example

Now we have defined the concept of an aggregated task at server j, we can establish the definitions for the following terms in order to understand the formulation presented next.

- **Transmission time for aggregated task at server j:** The time for server i to transmit its tasks(n_{ij}) to server j is n_{ij} divided by the link rate (B_{ij}). So the time it takes for all the tasks(aggregated tasks) assigned to server j is the sum of all servers' transmitting time, as shown in (1).

$$Transmission\ time\ at\ server\ j : \sum_{i=1}^{k} \frac{n_{ij}}{B_{ij}} \quad (1)$$

- **Turnaround time for aggregated task at server j:** The time it takes for server j to process all the tasks offloaded from other servers is calculated by taking the total CPU cycles required, $\sum_{i=1}^{k} n_{ij} * x$, divided by f_j, the CPU frequency of server j, as shown in (2).

$$\mathit{Turnaround\ time\ at\ server\ j}\ :\sum_{i=1}^{k}\frac{n_{ij}*x}{f_j} \tag{2}$$

- **Completion time for aggregated task at server j:** Completion time at server j is the time it takes to complete its aggregated task. It is the sum of transmission time (1) and turnaroud time (2), as shown in (3).

$$\mathit{Completetion\ time\ at\ server\ j}\ :\sum_{i=1}^{k}(\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j}) \tag{3}$$

- **Lateness for aggregated task at server j:** Lateness is defined by subtracting completion time (3) with D, the deadline, as shown in (4).

$$\mathit{Lateness\ at\ server\ j}\ :\left(\sum_{i=1}^{k}(\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j})\right)-D \tag{4}$$

B. Problem Formulation

The objective of the load balancing problem is to minimize deadline misses and total runtime. With the task model in place, we develop a mixed integer program for solving optimal load balancing of tasks across servers. The objective function is shown in (5). The optimization goal is to minimize deadline misses and total runtime. To analyze this objective function, first we recognize that $\left(\sum_{i=1}^{k}\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j}\right)-D$ is the lateness of completing aggregated tasks at server j, so minimizing this term reflects in minimizing total runtime. Secondly, $v*max\left(0,\sum_{i=1}^{k}\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j}\right)$ in (5) penalizes tardiness(missed deadlines) of aggregated task at server j, so minimizing this term reflects in minimizing deadline misses. Lastly v is used as weighing factor so one can decide to put more emphasis in minimizing deadline misses or total runtime. A more graphical analysis on the objective function (5) is shown in Figure 8.

$$\begin{aligned}Min:\ &\sum_{j=1}^{k}\Bigg[\left(\sum_{i=1}^{k}\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j}\right)\ -\ D\ +\\ &v*max\left(0,\sum_{i=1}^{k}\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j}\right)\Bigg] \qquad (5)\\ s.t.\ &\sum_{i=1}^{k}n_{ij}*x\le f_j*\tau \qquad for\ j\in\{1,2,..,k\} \qquad (6)\\ &\sum_{i=1}^{k}n_{ij}\le C_j \qquad for\ j\in\{1,2,..,k\} \qquad (7)\\ &\sum_{j=1}^{k}n_{ij}=N_i \qquad for\ i\in\{1,2,..,k\} \qquad (8)\\ &n_{ij}\in Z^{+} \qquad for\ i\in\{1,2,..,k\},\\ &\qquad\qquad for\ j\in\{1,2,..,k\} \qquad (9)\end{aligned}$$

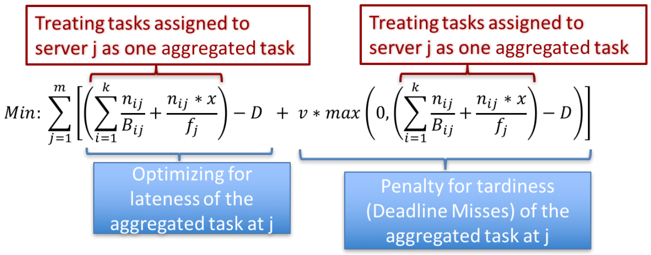

Treating tasks assigned to server j as one aggregated task
Treating tasks assigned to server j as one aggregated task
$Min: \sum_{j=1}^{m}\left[\left(\sum_{i=1}^{k}\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j}\right)-D\ +\ v*max\left(0,\left(\sum_{i=1}^{k}\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j}\right)-D\right)\right]$
Optimizing for lateness of the aggregated task at j
Penalty for tardiness (Deadline Misses) of the aggregated task at j

Figure 8: Analysis of Objective Function

The constraints for the optimization are shown from (6) to (9). Constraint (6) requires that every server has enough CPU cycles available for the tasks offloaded from other servers. Constraint (7) requires the total number of tasks allocated to a server does not exceed its capacity. Constraint (8) makes sure that all tasks will be distributed and processed. Constraint (9) makes sure a task will not be divided into smaller sub-tasks. Finally we reformulate the objective function (5) by introducing y_j as shown from (10) to (16) in order to solve the optimization using lp_solve[19].

$$\begin{aligned}Min:\ &\sum_{j=1}^{k}\left[\left(\sum_{i=1}^{k}\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j}\right)\ -\ D\ +v*y_j\right] \qquad (10)\\ s.t.\ &\sum_{i=1}^{k}n_{ij}*x\le f_j*\tau \qquad for\ j\in\{1,2,..,k\} \qquad (11)\\ &\sum_{i=1}^{k}n_{ij}\le C_j \qquad for\ j\in\{1,2,..,k\} \qquad (12)\\ &\sum_{j=1}^{k}n_{ij}=N_i \qquad for\ i\in\{1,2,..,k\} \qquad (13)\\ &n_{ij}\in Z^{+} \qquad for\ i\in\{1,2,..,k\},\\ &\qquad\qquad for\ j\in\{1,2,..,k\} \qquad (14)\\ &y_j\ge 0 \qquad for\ j\in\{1,2,..,k\} \qquad (15)\\ &y_j\ge\left(\sum_{i=1}^{k}\frac{n_{ij}}{B_{ij}}+\frac{n_{ij}*x}{f_j}\right)-D\\ &\qquad\qquad for\ j\in\{1,2,..,k\} \qquad (16)\end{aligned}$$

V. Results and Discussion

A. Experiment Setup

Our simulation adopts the system with seven local fog servers as illustrated in Figure 3 and an additional cloud server. For fog servers, the link rate between server i and server j, B_{ij}, is randomly drawn from between 16 and 64 tasks per second for $i\neq j$, or ∞ for $i=j$, the CPU frequency for server j, f_j, is randomly drawn from between 2700 MHz and 3600 MHz, and the capacity of server j, C_j, is randomly drawn from between 10 and 35 tasks. For the remote cloud server, its initial tasks, N_{cloud} is 0, B_{icloud} is 4 tasks per second, f_{cloud} is 4500MHz, and C_{cloud} is ∞. For all servers, CPU cycles required per task, x, is set at 35 Mcycles, deadline for each task, D, is set at 0.5 second, and allowed processing time for each server, τ, is 0.48 second. The system's parameters can be summarized in Table III.

A taskset is generated by randomly selecting a number for N_i from a range of integers which is shown in the first row in Table IV. By varying the range of integers for N_i, we can create different loadings for the system. In each experiment, for the same type of loading, taskset generation is performed 100 times, and load balancing optimization is executed for each newly generated taskset. We record all the N_i values and so each experiment can be run with the same tasksets. All the experiments were run in the machine with dual quadcore AMD Opteron 2.3 GHz with 16GB memory. The time it takes to execute linear prediction and to solve the optimization on that machine are both less than 0.01 second for all the experiments we run. So the overhead for mobility prediction and load balancing is less than 0.02 second, which is within the overhead budget, calculated by substracting deadline by allowed processing time, $D - \tau = 0.5 - 0.48 = 0.02$.

Table III: Simulation Parameters

Simulation Parameters for the System		
k	8	Number of servers
x	35	Number of CPU cycles required for each task (Mcycles)
D	0.5	Deadline for all tasks (sec)
τ	0.48	Allowed processing time for each server (sec)
Simulation Parameters for Local Fog Servers		
N_i	see Table IV	Number of initial tasks for server i
B_{ij}	[16,64]	Link rate between server i and server j for $i \neq j$ (tasks/sec)
	∞	Link rate between server i and server j for $i = j$ (tasks/sec)
C_j	[10,35]	Capacity of server j (tasks)
f_j	[2700,3600]	CPU frequency of server j (MHz)
Simulation Parameters for Cloud Server		
N_{cloud}	0	Number of inital tasks for cloud server
B_{icloud}	4	Link rate between server i and cloud server (tasks/sec)
C_{cloud}	∞	Capacity of cloud server (tasks)
f_{cloud}	4500	CPU frequency of cloud server (MHz)
Optimizing Variables		
n_{ij}	solved with optimization	Number of tasks distributed from server i to server j

Table IV: Types of Workloads

Range for N_i (tasks)	[10,26]	[10,28]	[10,30]	[10,32]	[10,34]
Total number of tasks	12687	13472	14143	14739	15051

B. Deadline Misses vs Total Runtime

In this experiment, we focus on the varying v, the weighing parameter in the optimization formulation presented. According to our objective function (10), a greater v value should result in fewer deadline misses with higher total runtime. We verify such trend by running experiment with different v and gather the results for deadline misses and total runtime respectively, as shown in Figure 9 and Figure 10. From the figures, we observe that a larger v does result in fewer deadline misses and higher total runtime.

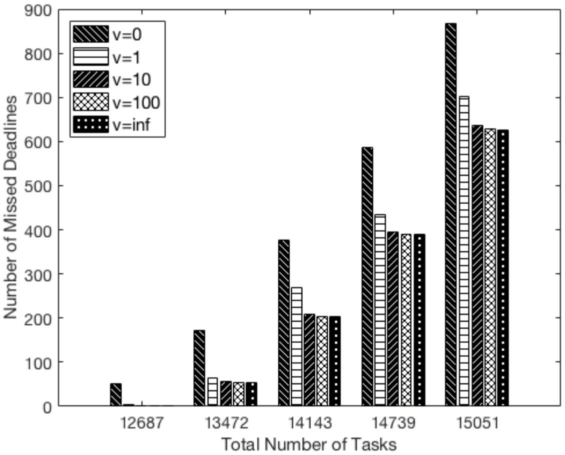


Figure 9: Deadline Misses Comparison with Different v

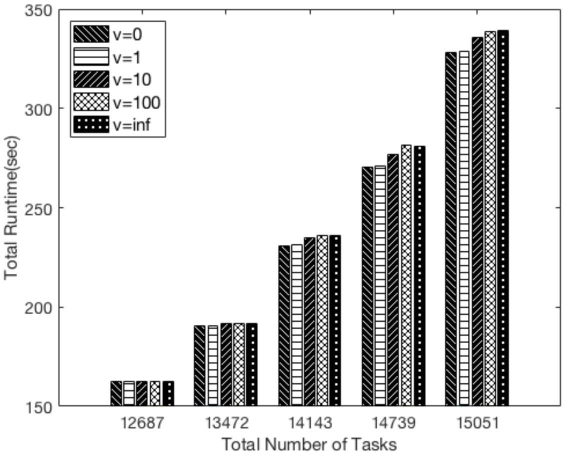


Figure 10: Total Runtime Comparison with Different v

C. Comparison with Other Load Balancing Algorithms

We compare our optimization result with local static, a simple load balancing method, and three other commonly used load balancing algorithms: weighted round robin, active monitoring and throttled load balancer[20][21][22]. In local static load balancing, each server with initial tasks exceed its capacity will transfer all the overloading tasks to the cloud server. Weighted round robin is implemented by assigning high weights for local servers that have not reached their maximum capacity. The cloud server will have a higher weight than a local server only when that server reaches its maximum capacity. Active monitoring load balancing is implemented by having each task assigned to the local server that has the highest remaining capacity, the algorithm will only assign to the cloud server when all the local servers are full. In the throttled load balancer, each task is assigned based on the most suitable server available, so the server with lowest completion time and has not reached its maximum capacity will be assigned with the new task. Again, a cloud server will be utilized only when none of the local servers are available. The comparisons are shown in Figure 11 and Figure 12. Local static does not utilize any available local server so it is not surprising that local static is outperformed by more than 50% in every case. On the other hand, our optimization has the least deadline misses and total runtime for every case. The numerical results for deadline misses is presented in Table V. From Table V, at a lighter load of 13472 tasks, our optimization outperforms throttled load balancer which has the second least deadline misses, by almost 50%, but as the load of the system increases, the improvement brought by optimization starts to decrease. At the heaviest load of 15051 tasks, the improvment our optimization can bring drops to about 25%. The reason being a heavy loaded system has fewer available options for task distributing, thus the potential for a better performance gets smaller as more load is added to the system.

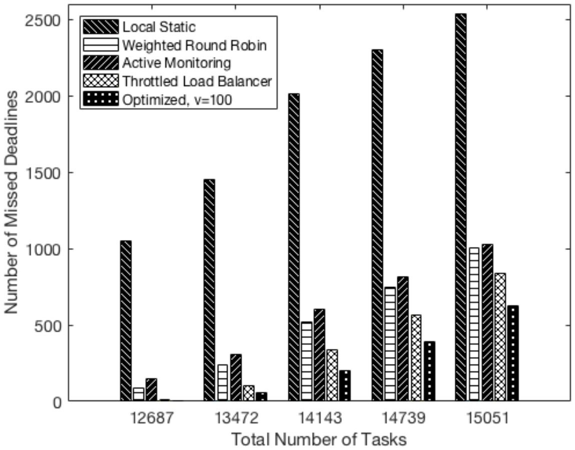


Figure 11: Deadline Misses Comparison with Other Load Balancing Algorithms

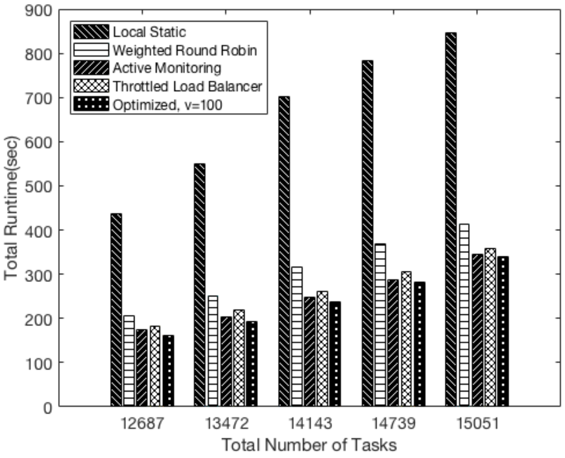


Figure 12: Total Runtime Comparison with Other Load Balancing Algorithms

Table V: Missed Deadlines Counts

Total number of tasks	**12687**	**13472**	**14143**	**14739**	**15051**
Optimized, v=100	0	54	204	389	627
Local Static	1052	1454	2011	2300	2535
Weighted Round Robin	85	240	517	745	1003
Active Monitoring	147	306	606	815	1026
Throttled Load Balancer	11	104	341	568	837

VI. Conclusion and Future Work

In this work, we propose a task scheduling model for connected car system in fog computing that brings task scheduling from device level to server level, which reduces the amount of computations required to perform load balancing. We also formulate a load balancing optimization problem that minimizes deadline misses and total runtime. We show that our optimization outperforms some common load balancing algorithms such as weighted round robin, active monitoring, and throttled load balancer. There are many potentials and ideas to expand on this topic including take energy consumption into consideration, vary the parameters dynamically, or implement applications with real workloads. We look forward to pursuing more research in this topic as the development of fog computing and connected car systems continues.

References

[1] 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, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2017.

[3] K. Kai, W. Cong, and L. Tao, "Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues," *The Journal of China Universities of Posts and Telecommunications*, vol. 23, no. 2, pp. 56 – 96, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1005888516600213

[4] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane, "Software defined networking-based vehicular adhoc network with fog computing," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 1202–1207.

[5] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, "Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions," *IEEE Communications Surveys Tutorials*, vol. 13, no. 4, pp. 584–616, Fourth 2011.

[6] J. Oueis, E. C. Strinati, and S. Barbarossa, "The fog balancing: Load distribution for small cell cloud computing," in *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, May 2015, pp. 1–6.

[7] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, Dec 2016.

[8] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, ser. MCC '13. New York, NY, USA: ACM, 2013, pp. 15–20. [Online]. Available: http://doi.acm.org/10.1145/2491266.2491270

[9] T. Nishio, R. Shinkuma, T. Takahashi, and N. B. Mandayam, "Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud," in *Proceedings of the First International Workshop on Mobile Cloud Computing &#38; Networking*, ser. MobileCloud '13. New York, NY, USA: ACM, 2013, pp. 19–26. [Online]. Available: http://doi.acm.org/10.1145/2492348.2492354

[10] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Opportunistic spatio-temporal event processing for mobile situation awareness," in *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, ser. DEBS '13. New York, NY, USA: ACM, 2013, pp. 195–206. [Online]. Available: http://doi.acm.org/10.1145/2488222.2488266

[11] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[12] L. Chunlin, L. Yanpei, and L. Youlong, "Energy-aware cross-layer resource allocation in mobile cloud," *International Journal of Communication Systems*, vol. 30, no. 12, pp. e3258–n/a, 2017, e3258 IJCS-16-0378.R1. [Online]. Available: http://dx.doi.org/10.1002/dac.3258

[13] P. Salvador and A. Nogueira, "Markov modulated bivariate gaussian processes for mobility modeling and location prediction," in *Proceedings of the 10th International IFIP TC 6 Conference on Networking - Volume Part I*, ser. NETWORKING'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 227–240. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008780.2008803

[14] D. Stynes, K. N. Brown, and C. J. Sreenan, "A probabilistic approach to user mobility prediction for wireless services," in *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, Sept 2016, pp. 120–125.

[15] "Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments," *IEEE Std 802.11p-2010 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009)*, pp. 1–51, July 2010.

[16] J. Gozalvez, M. Sepulcre, and R. Bauza, "Ieee 802.11p vehicle to infrastructure communications in urban environments," *IEEE Communications Magazine*, vol. 50, no. 5, pp. 176–183, May 2012.

[17] K. B. Baltzis, "Hexagonal vs circular cell shape: a comparative analysis and evaluation of the two popular modeling approximations," in *Cellular Networks-Positioning, Performance Analysis, Reliability*. InTech, 2011.

[18] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi, "CRAWDAD dataset roma/taxi (v. 2014-07-17)," Downloaded from http://crawdad.org/roma/taxi/20140717, Jul. 2014.

[19] M. Berkelaar, K. Eikland, P. Notebaert *et al.*, "lpsolve: Open source (mixed-integer) linear programming system," *Eindhoven U. of Technology*, 2004.

[20] P. Samal and P. Mishra, "Analysis of variants in round robin algorithms for load balancing in cloud computing," *International Journal of computer science and Information Technologies*, vol. 4, no. 3, pp. 416–419, 2013.

[21] S. G. Domanal and G. R. M. Reddy, "Load balancing in cloud computingusing modified throttled algorithm," in *2013 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Oct 2013, pp. 1–5.

[22] M. Sharma and P. Sharma, "Performance evaluation of adaptive virtual machine load balancing algorithm," *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 3, no. 2, 2012.