Towards an Adaptive Multi-modal Traffic Analytics Framework at the Edge

Geoffrey Pettet, Saroj Sahoo, Abhishek Dubey

Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, USA

***Abstract*—The Internet of Things (IoT) requires distributed, large scale data collection via geographically distributed devices. While IoT devices typically send data to the cloud for processing, this is problematic for bandwidth constrained applications. Fog and edge computing (processing data near where it is gathered, and sending only results to the cloud) has become more popular, as it lowers network overhead and latency. Edge computing often uses devices with low computational capacity, therefore service frameworks and middleware are needed to efficiently compose services. While many frameworks use a top-down perspective, quality of service is an emergent property of the entire system and often requires a bottom up approach. We define services as multi-modal, allowing resource and performance tradeoffs. Different modes can be composed to meet an application's high level goal, which is modeled as a function. We examine a case study for counting vehicle traffic through intersections in Nashville. We apply object detection and tracking to video of the intersection, which must be performed at the edge due to privacy and bandwidth constraints. We explore the hardware and software architectures, and identify the various modes. This paper lays the foundation to formulate the online optimization problem presented by the system which makes tradeoffs between the quantity of services and their quality constrained by available resources.**

I. Introduction

Emerging Trends: With the burst of the cloud computing paradigm in the last decade and a half, systems requiring intensive computations over large data volumes have relied on shared data centers to which they transfer their data for processing. However, traditional cloud computing architecture is problematic in a number of application domains that are either latency sensitive or cannot transfer data across the backhaul due to bandwidth constraints or privacy concerns. To alleviate these situations, engineers have leveraged the computing power of nearby available resources, leading to a profound discussion on the opportunistic usage of the computing resources dispersed in the community. In cases where the resources are not available near the physical edge, people have begun to add low power computing devices. These edge devices are then responsible for processing and sending data to a central server. Some examples of edge systems are the SCALE-2 [1] platform which runs air-quality monitoring sensors, collections of edge devices that provide computing services with low latency closer to the physical processes [2], and the Paradrop architecture [3] which provides the capability to run containerized applications in network routers.

Application of Interest: Finding anomalous traffic patterns is called traffic anomaly detection, and is critical to helping city planners optimize urban transportation systems and reduce congestion. Many data and model driven methods have been proposed to identify faulty sensors [4], [5], [6] Some researchers [7], [8], [9] have worked on detecting traffic events such as car accidents and congestion using videos, traffic, and vehicular ad-hoc data, while others have have explored the root causes of anomalous traffic [10], [11], [12], [13].

Most existing work still focuses on individual road sections or small regions to identify traffic congestion, but few studies explore non-recurring congestion (NRC) and its causes for a large urban area. Recently, deep learning techniques have achieved success in research fields including image processing, speech recognition, and bioinformatics. They provide an opportunity to potentially solve the NRC identification and classification problem. However, the state of the art is still to collate the data into a server and then perform the NRC classification periodically; Mobile Edge and Fog Computing provides a new opportunity.

Challenges: Complex event processing frameworks and query processing middleware like Internet Flow of things [14] primarily focus on temporary service composition and data query aggregation from the top, but often applications require a bottom-up design perspective. It is well known that, in general, quality of service is an emergent property of the entire system. The challenge is to understand how to construct a system (collection of applications) from software components such that the system-level properties can be established by reasoning about the composition at runtime. However, this is difficult considering that prior research in distributed application composition and resource allocation has often assumed global, reasonably accurate, and trusted knowledge of resource states or types [15], smaller optimization problem scale [16], limited churn in resource availability, and coordinated communication [17]. These assumptions severely restrict the ability to realize the full potential of the fog computing vision, which requires the ability to dynamically track, discover, and compose available resources at ultra-large scales to ensure the optimal set of objectives, graded by criticality, are met, resources are not over-committed, and critical constraints are adhered to despite limited and unreliable communication between the entities requesting and using resources. Therefore, new uncertainty-aware algorithmic approaches are required.

One way to handle uncertainty is to make each of the actors **multi-modal**. By multi-modal we refer to different operating modes of the actor, wherein each mode provides the same interface and service, but differs in the level of algorithm complexity. Consequently, we can trade-off higher performance for

lower resource consumption. For instance, a traffic analyzer actor can use a less computationally expensive Background Subtraction Algorithm when the traffic is expected to be low, and transition to computationally expensive deep learning based object classification [18] algorithms in peak hours of traffic. Once different modes and their performance based on context and available resources are known, we can set up an online optimization problem which describes the tradeoff between the computational performance and availability of resources, represented using a Pareto front.

Contributions: In this paper we describe the design of a multi-modal traffic analytics application deployed at the edge. We further describe its multi-modal configuration in terms of different variants available to be used. We then discuss the constraints on a real world deployment of such a system and how those constraints affect hardware and software decisions. Our approach is inspired by our earlier approach for modeling an application's goals as a high-level function, which are then decomposed into smaller sub-functions [15]. The forest of trees thus created describes the various functions in the systems. The functions at the leaf level are mapped to specific components that can be used to implement that function. Then, given the information about the context description and the constraints related to component composition and the requirements imposed by the components, application deployment, failure avoidance, fault management, and operations management of distributed systems can be automated as shown by our prior work on the CHARIOT platform [15], [19] by encoding the constraints in the form of a matrix of decision variables. By decomposing the services themselves into modes with various performance and resource requirement trade-offs, the configuration problem defined by CHARIOT can be extended to include the mode selection and configuration to optimize for performance as well as resiliency and feasibility. This can be extended to to work for any edge service that has modes with similar performance trade-offs.

Outline: In section II we describe the problem and its constraints. In section III we then describe the hardware and software architecture we deployed, the detection techniques available, as well as detail why we made decisions based on the given constraints. We then detail experiments run on the system to compare the performance of the tested models for traffic detection in Section IV. Last we offer concluding remarks in section VI.

II. The Traffic Analytics Application

This paper describes a block level traffic sensing architecture. At each block we provide a set of cameras with overlapping views and a set of edge computing devices, specifically raspberry pis and Nvidia Jetson TX2s. Cameras are easy to install and use, and are flexible in that the same cameras can be used for vehicles, pedestrians, and parking sensing in the future. Using these devices we implement the hierarchical traffic analytics workflow as shown in figure 1. Key components in this architecture are the algorithms to detect and track cars and generate the count and density information per area of interest. Note that in this paper we focus on the primary features required to detect non-recurring congestion, as the specific non-recurring congestion algorithm has been described in our prior work [20].

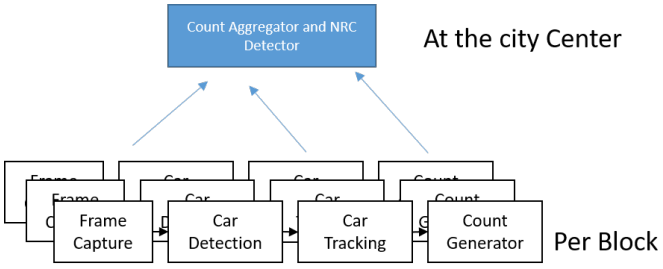

Fig. 1. The traffic analytics workflow

A. Constraints

Next we discuss the constraints we considered when we deployed the analytics workflow in Nashville.

Network Limitations: Initially devices were connected to the cloud using a single wireless connection, which presents a bottleneck: high bandwidth image and video data cannot be sent over the network reliably. This implies that image processing should occur at the edge to avoid network congestion. Recently the city ran fiber cable to the intersections, so the bandwidth issue is less relevant for the pilot installation. However, Nashville is interested in scaling to many intersections in the city, which means that the system must be able to run on the city's many wirelessy connected intersections. Therefore, we have to consider hierarchical block-level edge computing solutions.

Privacy: Government regulations in Nashville require that the images captured and used by the system must be kept on a private network: no identifiable information can be sent over insecure channels or kept for long periods of time, as it could potentially leak and cause privacy issues for citizens. While it is possible to encrypt the data, it would place a large computational toll on the embedded system. Practically, this reinforces that the images or video must be processed on the edge rather than being sent to a cloud.

Resilience: Installation of such a system can be quite expensive, and the city has few resources to devote to maintaining it. The hardware will be running 24/7, with the cameras out in the elements, so it is expected that there will be some hardware failures. Therefore the system needs to be able to tolerate some level of failure without needing physical intervention.

III. Implementation Architecture

A. Software Component Modes

We now discuss the algorithms we chose as modes in our traffic analytics workflow shown in figure 1.

Open CV: Image Object detection has been studied for several years. Methods have evolved from relatively simple techniques such as Background subtraction [21], Haar Cascades [22], and Dense Optical Flow [23]. These techniques are all freely available in OpenCV, an open source computer vision library [24]. A large breakthrough in accuracy occurred

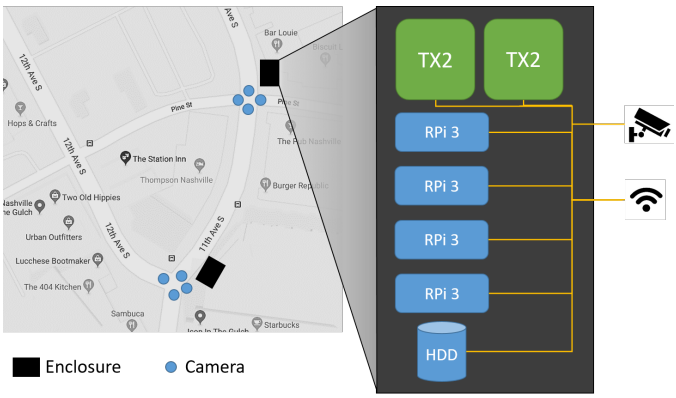

Fig. 2. Hardware Diagram

with the application of deep learning to the object detection problem [25], [26], making these techniques relatively obsolete [27], [28].

Faster Recurrent Convolutional Neural Network (Faster-RCNN): this is an evolution of one of the the first frameworks to use convolutional neural networks (CNNs) for object detection, R-CNN [29]. It uses a region proposal network (RPN) to find regions likely to contain objects, uses a base network like VGG-16 [30] or Resnet-101 [31] for feature extraction, and a 3x3 sliding window that moves across the feature map. These region proposals are then fed into a CNN for object detection. This multi-step process improves accuracy at a cost of computation.

Single Shot detector Multibox Detector (SSD): SSD [32] does the task of object localization and classification in a "single shot", i.e. a single forward pass of the network. The implementation using MobileNet [33], a lightweight network, is targeted for real time processing on resource constrained devices.

YOLO v3: This [34] is an evolution of the You Only Look Once detection framework [35]. Unlike RPNs, it takes the input image and divides it into grids. For each grid it predicts class probabilities, bounding boxes, and the box's confidence scores. It uses 2 stacked 53 layer networks, making a 106 layer fully convolutional network, and makes detections at 3 different scales.

SORT (Simple Online and Realtime Tracking): Once an object is detected, it is imperative to track it between frames to avoid double counting. SORT [36] tracks objects in real time using a combination of techniques such as Kalman Filters [37] and the Hungarian Algorithm [38]. It assigns each detection in the frame an id that is persistent across frames. Data association is done using the IOU (intersection over union) distance metric between neighbouring frames. It uses a simple constant velocity model to predict the dynamics of objects if detections are lost, thus the quality of tracking is highly dependent on the quality of object detection algorithm.

B. Hardware Components

Figure 2 shows the hardware layout at two intersections in the city. The hardware used in the system includes 4 cameras, 4 Raspberry Pi 3s, 2 Nvidia Jetson TX2s, and a 1 terabyte hard drive for each intersection's cluster of devices. In addition, our system has access to a cloud server for storing and accessing the metrics produced. The specifics of the physical components are described below.

Cameras: We installed four cameras at each intersection, ensuring that there is partial overlap between their views. We used Reolink's 410s models (https://reolink.com/product/rlc-410s/), which use Power over Ethernet (PoE). They were mounted to traffic poles over the intersection to allow access to unobstructed views of traffic. While this gives flexibility in how views are chosen to maximize the performance of the detection algorithms, it comes at the cost of expensive installation, which involves a costly process including obtaining road closing permits, police to direct traffic, and the contraction of an installation company. It is important to ensure that we decouple the camera from computing resources, which allows the compute device to still be used even when cameras fail.

GPU Resources: Each intersection has two Nvidia TX2s, which serve as the primary object detection devices of each cluster. The detection techniques we consider use deep learning architectures, which run significantly faster on dedicated GPUs than on the CPUs of typical embedded devices [39]. Nvidia's Jetson systems integrate a GPU with 256 cuda cores with a low power, ARM CPU. Due to the high cost compared to other components of the system, we are limited on the number of TX2s, and deploy only two per intersection. Two devices allows for redundancy in case one fails at an intersection.

Primary Compute Device: Each intersection has four Raspberry Pi 3s. These are inexpensive, relatively reliable, and efficient, making them ideal edge computation devices. They serve to remove load from the TX2s by connecting to the cameras to pull video, running a cluster-local database to backup metrics, and running any computational algorithms that do not require the GPU.

Storage: Each cluster has a one terabyte hard disk, connected to one of the Raspberry Pis, that stores a cyclic database of metrics output by the system.

C. Implemented Workflow

Figure 4 shows an overview of the system's software architecture, which is composed of several connected components. Each service is described below, and the source code can be found at https://github.com/gap101/EdgeTrafficAnalytics.

Camera Service: This service manages a particular camera feed, and uses Real-Time Messaging Protocol (RTMP) to receive new frames from the camera. We are able to maintain about 9 frames per second over our network. We keep the service running even if the camera feed isn't being actively used. The service receives frames from the camera, and serves them to any number of connected object detection services.

Object Detection: The first step of image processing is to get good vehicle detections in frames of the videos. We use modern machine learning based approaches using the Jetson TX2s. We settled on the three architectures for vehicle detection detailed in section II: YOLO v3, Faster RCNN, and SSD. We hypothesized that these models have various computation vs accuracy trade offs: for example, Faster-RCNN

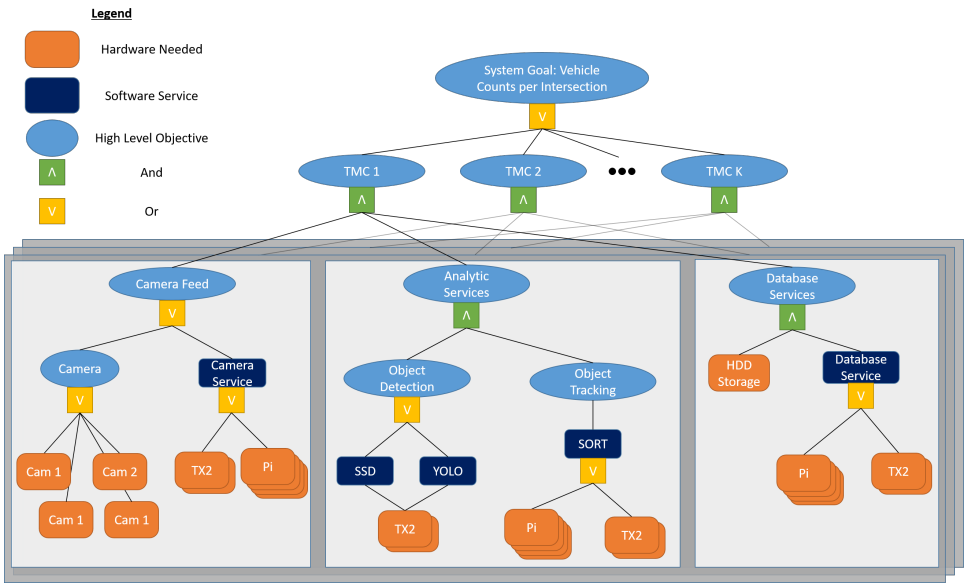

Fig. 3. System Components and Goals. These specification is compiled down to a SMT problem which provide runtime adaptation. In our ongoing work we are working on developing a quality of service based optimization approach to adaptation.

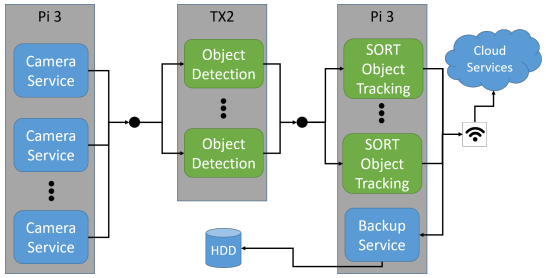

Fig. 4. System Software Architecture Overview. The flow of data through the system and the devices each service are run on are shown.

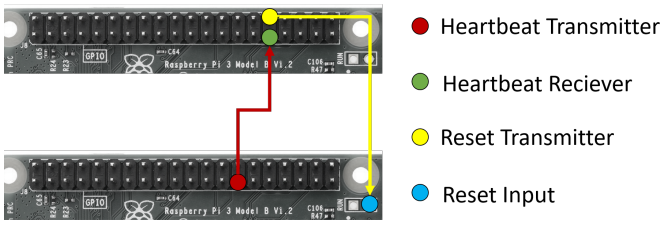

Fig. 5. Diagram of the GPIO pins used in the Pi's watchdog service. Peroidic pulses are sent via the heartbeat pin. If they are not detected for 10 seconds, a reset signal is sent

is a larger and slower network than SDD, but is more accurate [40]. We examine the specific tradeoffs in section IV.

Object Tracking and Counting: Once vehicles are detected, their bounding boxes are sent to SORT for tracking between frames. Unfortunately the object detectors will occasionally miss an object for a few frames, causing SORT to re-identify it. To address this, we considered a trapezoidal region of interest (ROI) in the center of the intersection that all cars pass through. Vehicles are counted when they pass through the ROI, avoiding false detections and double counting.

Local Cyclic Storage: This service connects to a local MongoDB cyclic database, which is used to backup the system's output in case of network or other unforeseen failures on the server end. It runs on the Pi connected to the HDD.

Cloud Services: Once an image is processed, the resulting metrics are sent to a cloud server, which stores the results in a InfluxDB instance. There is also a dashboard built using Grafana that allows users to inspect the data in real time.

D. Making the architecture resilient

We designed the architecture to tolerate hardware and sofware failures. Figure 3 describes the goals of the system along with the software and hardware components that are needed. The high level goal of the system is to count vehicles that go through various intersections where hardware is deployed. The rest of the diagram describes how that goal can be reached, and can be treated as a SMT problem [19]. As long as the constructed SMT problem is satisfied, i.e. the required hardware with required resources is available, the goal can be met. For example, by ensuring that at least 3 cameras have overlapping views of the intersection we can tolerate failures of two cameras without compromising data acquisition.

Further, to prevent the lockup of computing devices we set up a hardware watchdog service using the GPIO pins on the compute devices. Each device sends a heartbeat pulse every second on one of its pins, which is watched by other devices in the cluster. If the pulse is not detected for 10 seconds, it is assumed that the device has frozen, and a hardware reset signal is sent. A diagram of the watchdog connections between two of the pis is shown in figure 5.

Lastly, we run all analytic services on the nodes as systemd scripts which are set to restart upon failure and reboot. Thus, the service fails only when the computing device itself fails. To address this, we defined a flexible software architecture that allows the services to be easily moved between devices. When a device fails, the services can be moved to any device of the same type (so object detection running on a TX2 can be moved to another TX2, for example) using similar reconfiguration

TABLE I
PENALIZED ACCURACY OF MODELS. THREE 3 MINUTE LONG VIDEOS WERE RAN THROUGH EACH MODEL (PLUS SORT) AT EACH TIME OF DAY, AND THE ACCURACY OF THE MODEL IS AVERAGED ACROSS THE VIDEOS. THE PENALIZED ACCURACY IS DEFINED AS THE RATIO: ((NUMBER OF CARS CORRECTLY COUNTED) - (NUMBER OF CARS INCORRECTLY COUNTED)) / (ACTUAL CAR COUNT).

Time of Day	SSD+SORT	YOLO+SORT	F-RCNN+SORT
04:00	0.25	.66	.25
08:00	0.60	0.90	0.89
17:00	0.75	0.99	0.95
22:00	0.27	0.91	0.89
Avg Accuracy	**0.61**	**0.95**	**0.93**
Avg Errors/Min	**12.6/min**	**1.83/min**	**2.75/min**

TABLE II
AVERAGE INFERENCE TIME FOR EACH MODEL PER FRAME IN SECONDS.

SSD	YOLO	Faster RCNN
0.29 sec	0.62 sec	1.17 sec

strategies explored in our previous CHARIOT system [19].

IV. QUANTIFYING PERFORMANCE AND RESOURCE REQUIREMENTS

To understand how we can choose one possible analytical workflow from the available alternatives (Yolov3 + SORT, SSD + SORT and F-RCNN + SORT), we compared them across several dimensions: penalized accuracy, computation time, and power draw. We experimented using 3 minute videos at four different times of day: 03:00, 08:00, 17:00, 22:00. These times cover different lighting and traffic conditions: late night with little traffic, early morning with medium traffic, evening with heavy traffic, and night with medium traffic.

We define the penalized accuracy of the models as $A_p = \frac{\text{\# cars correctly counted} \quad \text{\# cars incorrectly counted}}{\text{Actual \# cars}}$, which accounts for double counting cars. The results are shown in table I. First, an obvious observation is that YOLO and Faster RCNN are fairly similar, with YOLO being slightly more accurate at 95% vs 93% average accuracy. SSD performs significantly worse, at 61%.

Another observation is that all of the algorithms perform significantly worse at 04:00 than other times, which is due to blurry images caused by long shutter speeds at night. This isn't as pronounced at 22:00, as traffic is still dense and cars move slowly, minimizing blur. Further, SSD's performance degrades at 22:00, while the other algorithms remain fairly accurate, which implies that SSD is not well suited for running in the dark.

We examined the average computation time needed on the TX2 to process a frame of video using each of the model. The average inference time per frame for SSD was $0.29sec$, YOLO took $0.62sec$ and RCNN took $1.17sec$. None are fast enough for real time tracking, so we must process the data in batches. Lastly, we examined the power consumed by the TX2 while running each of the models. While YOLO ($13.5watts$) and Faster RCNN ($13.9watts$) use similar amounts of power, SSD uses nearly 30% less ($9.6watts$).

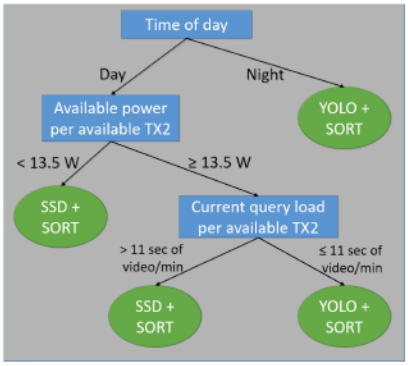

Fig. 6. Mode Selection Decision Tree. Note that these are optimal choices and upon failures we can select alternatives using the goal tree shown in figure 3.

V. DISCUSSION

F-RCNN is not worth considering for deployment, since its performance and accuracy is worse than YOLO. YOLO takes twice as long as for inference as SSD, but has significantly higher accuracy. Therefore YOLO is preferred unless resources fail or become constrained due to a high query load. Cameras produce 9 frames of video per second, so based off the inference time in table II, each TX2 running YOLO can process 11 seconds of video each minute, while SSD can process 23 seconds of video each minute.

SSD has a steep drop in accuracy at night. Therefore at night YOLO is preferred, even if the sampling rate must be decreased.

We derived the decision tree for which object detector to use in figure 6. At night, YOLO is used due to SDD's unacceptable accuracy. Otherwise, YOLO is preferred, but the system will switch to SSD if forced by either a power or query load constraint. We use this decision tree in conjunction with CHARIOT [19] to choose the appropriate configuration at runtime. The decision tree is used to selectively add constraints to guide the feasibility engine towards a specific set of components. When that set of components infeasible, we search for the feasibility of the next best choice and so on. Note that in future rather than encoding a decision order we will set this problem as a stochastic optimization problem.

VI. CONCLUSION

We have examined and implemented a complete IoT architecture that counts vehicles going through intersections. Our decisions and constraints are detailed so that readers can easily apply them to their own systems. The system uses multi-modal sub components, which allows for different configurations depending on resource availability and query demand. Unlike edge system management tools such as CHARIOT [19], this allows for component reconfiguration based on optimizing performance as well as recovering from failure. We currently apply a static mode switching function, but we have laid the foundation for an online optimization problem which describes the trade-off between the computational performance and availability of resources. This architecture can be extended to work with any edge service that has sub-modes with such a trade-off.

Acknowledgement:

This work is sponsored in part by the National Science Foundation under the award numbers CNS-1647015 and CNS-1818901. We thank our partners in the Metro Nashville government.

REFERENCES

[1] K. Benson, C. Fracchia, G. Wang, Q. Zhu, S. Almomen, J. Cohn, L. D'arcy, D. Hoffman, M. Makai, J. Stamatakis, and N. Venkatasubramanian, "Scale: Safe community awareness and alerting leveraging the internet of things," *IEEE Communications Magazine*, vol. 53, no. 12, pp. 27–34, Dec 2015.

[2] M. García-Valls, A. Dubey, and V. Botti, "Introducing the new paradigm of social dispersed computing: Applications, technologies and challenges," *Journal of Systems Architecture*, vol. 91, pp. 83 – 102, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762118301036

[3] D. Willis, A. Dasgupta, and S. Banerjee, "ParaDrop: A Multi-tenant Platform to Dynamically Install Third Party Services on Wireless Gateways," in *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*. ACM, 2014, pp. 43–48.

[4] X.-Y. Lu, P. Varaiya, R. Horowitz, and J. Palen, "Faulty loop data analysis/correction and loop fault detection," in *15th World Congress on Intelligent Transport Systems and ITS America's 2008 Annual Meeting*, 2008.

[5] N. Zygouras, N. Panagiotou, N. Zacheilas, I. Boutsis, V. Kalogeraki, I. Katakis, and D. Gunopulos, "Towards detection of faulty traffic sensors in real-time." in *MUD@ ICML*, 2015, pp. 53–62.

[6] A. Ghafouri, A. Laszka, A. Dubey, and X. Koutsoukos, "Optimal detection of faulty traffic sensors used in route planning," in *Proceedings of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering*. ACM, 2017, pp. 1–6.

[7] S. Kamijo, Y. Matsushita, K. Ikeuchi, and M. Sakauchi, "Traffic monitoring and accident detection at intersections," *IEEE transactions on Intelligent transportation systems*, vol. 1, no. 2, pp. 108–118, 2000.

[8] S. Yang, K. Kalpakis, and A. Biem, "Detecting road traffic events by coupling multiple timeseries with a nonparametric bayesian method," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 1936–1946, 2014.

[9] X. Kong, X. Song, F. Xia, H. Guo, J. Wang, and A. Tolba, "Lotad: long-term traffic anomaly detection based on crowdsourced bus trajectory data," *World Wide Web*, pp. 1–23, 2017.

[10] L. Xu, Y. Yue, and Q. Li, "Identifying urban traffic congestion pattern from historical floating car data," *Procedia-Social and Behavioral Sciences*, vol. 96, pp. 2084–2095, 2013.

[11] A. H. Chow, A. Santacreu, I. Tsapakis, G. Tanasaranond, and T. Cheng, "Empirical assessment of urban traffic congestion," *Journal of advanced transportation*, vol. 48, no. 8, pp. 1000–1016, 2014.

[12] S. Kwoczek, S. Di Martino, and W. Nejdl, "Stuck around the stadium? an approach to identify road segments affected by planned special events," in *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*. IEEE, 2015, pp. 1255–1260.

[13] R. Al Mallah, A. Quintero, and B. Farooq, "Distributed classification of urban congestion using vanet," *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[14] K. Yasumoto, H. Yamaguchi, and H. Shigeno, "Survey of real-time processing technologies of iot data streams," *Journal of Information Processing*, vol. 24, no. 2, pp. 195–202, 2016.

[15] S. Pradhan, A. Dubey, T. Levendovszky, P. S. Kumar, W. A. Emfinger, D. Balasubramanian, W. Otte, and G. Karsai, "Achieving resilience in distributed software systems via self-reconfiguration," *Journal of Systems and Software*, vol. 122, pp. 344 – 363, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121216300590

[16] Y. Sun, J. White, B. Li, M. Walker, and H. Turner, "Automated qos-oriented cloud resource optimization using containers," *Automated Software Engineering*, vol. 24, no. 1, pp. 101–137, Mar 2017.

[17] N. Roy, A. Dubey, A. Gokhale, and L. Dowdy, "A capacity planning process for performance assurance of component-based distributed systems," in *Proceedings of the 2Nd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '11. New York, NY, USA: ACM, 2011, pp. 259–270. [Online]. Available: http://doi.acm.org/10.1145/1958746.1958784

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[19] S. Pradhan, A. Dubey, S. Khare, S. Nannapaneni, A. Gokhale, S. Mahadevan, D. C. Schmidt, and M. Lehofer, "Chariot: Goal-driven orchestration middleware for resilient iot systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 2, no. 3, pp. 16:1–16:37, Jun. 2018. [Online]. Available: http://doi.acm.org/10.1145/3134844

[20] F. Sun, A. Dubey, and J. White, "Dxnat—deep neural networks for explaining non-recurring traffic congestion," in *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2141–2150.

[21] M. Piccardi, "Background subtraction techniques: a review," in *Systems, man and cybernetics, 2004 IEEE international conference on*, vol. 4. IEEE, 2004, pp. 3099–3104.

[22] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–I.

[23] A. Talukder and L. Matthies, "Real-time detection of moving objects from moving vehicles using dense stereo and optical flow," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 4. IEEE, 2004, pp. 3718–3725.

[24] O. Team. Opencv. [Online]. Available: https://opencv.org/

[25] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[27] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[28] J. Han, D. Zhang, G. Cheng, N. Liu, and D. Xu, "Advanced deep-learning techniques for salient and category-specific object detection: a survey," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 84–100, 2018.

[29] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[32] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[33] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[34] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[36] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *Image Processing (ICIP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 3464–3468.

[37] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[38] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[39] A. Lazorenko. (2017) Tensorflow performance test: Cpu vs gpu. [Online]. Available: https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c

[40] J. Hui. (2018) Object detection: speed and accuracy comparison (faster r-cnn, r-fcn, ssd, fpn, retinanet and yolov3). [Online]. Available: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359