A Cloud-Based Microservices Solution for Multi-Agent Traffic Control Systems

Chikadibia Ihejimba
The University of Texas at Dallas
Richardson, United States
cki100020@utdallas.edu

Rym Z. Wenkstern
The University of Texas at Dallas
Richardson, United States
rymw@utdallas.edu

ABSTRACT

In this paper, we present MATS-Cloud, a cloud-based microservices architecture designed for multi-agent-based Traffic Control Systems (TCS). This architecture facilitates decentralized control, real-time responsiveness, and dynamic scalability. To validate our approach, we have transitioned *DALI*, a multi-agent collaborative TCS, from its current server-based architecture to the cloud, implementing this new architecture. Experimental results demonstrate significant improvements in latency reduction and system adaptability when compared to the traditional server-based model.

KEYWORDS

Multi-Agent Traffic Control Systems; Intelligent Transportation Systems; Cloud-Native; Microservices; Serverless

ACM Reference Format:
Chikadibia Ihejimba and Rym Z. Wenkstern. 2024. A Cloud-Based Microservices Solution for Multi-Agent Traffic Control Systems. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

Traffic congestion is a rising concern in urban areas across the globe, resulting in longer commute times, decreased productivity, and increased air pollution due to greenhouse gas emissions from idling vehicles. As a result, the development of effective Traffic Control Systems (TCS) has gained increasing significance, aligning with the Sustainable Development Goals (SDGs) [5, 22], particularly SDG 11, which underscores the importance of sustainable urbanization and the promotion of safe, inclusive, and resilient cities.

Over the years, the literature has seen numerous papers addressing agent-based Traffic Control Systems [1–4, 7, 13, 20, 21, 24, 39, 41]. However, the primary emphasis in Multi-Agent Traffic control System (MATS) research has been on traffic control algorithms, often neglecting the challenges associated with deployment. As a result, due to the extremely limited number of real-world implementations of MATS, most traffic lights are still managed by intersection controllers directly or indirectly connected to a higher-level central traffic management unit [26, 30, 37, 38]. In the US, the transportation community has recognized the much-needed concepts of distribution, intelligence, autonomy, and collaboration, but only a few very MATSs have been successfully deployed [32, 36], and their implementations have predominantly utilized conventional server-based architectures.

This paper introduces MATS-Cloud, a novel cloud-native architecture for multi-agent TCS. We present MATS-Cloud's architecture and discuss its implementation through *DALI* [33–36], a multi-agent, collaborative traffic control system currently operating in the US. In *DALI*, an agent that serves as the 'brain' of a controller executes in a server-based infrastructure and connects to its controller via VPN, all while maintaining constant communication with other agents through direct links. With the loose coupling design approach adopted for *DALI*, agents can be deployed on various devices (e.g., PC, laptop, cell phone, Raspberry Pie) and run either remotely or be seamlessly integrated into the controller.

Currently, MATS-Cloud's implementation leverages Amazon Web Services (AWS), a renowned leader and pioneer in cloud-native infrastructure solutions. It incorporates the latest cloud technologies, including cloud-managed serverless computing, Docker containers, and software-defined networking solutions. Additionally, it employs a serverless relational database with geo-distributed read/write clusters and auto-scalability built-in. Experimental results demonstrate that our cloud-native solution operates smoothly, scales efficiently, and exhibits lower latency compared to the conventional server-based solution.

The rest of this paper is organized as follows: in Section 2, we discuss related works and highlight the unique contributions of our work; in Section 3, we describe the system design, and in Section 4, we evaluate MATS-Cloud for latency and scalability.

2 RELATED WORKS

The taxonomy of existing TCSs includes two major categories: Non-MAS and MAS, and can also be further classified as either non-cloud-based or cloud-based. Furthermore, cloud-based solutions can be categorized as non-cloud-native or cloud-native.

A plethora of research-based solutions for TCS algorithms have been proposed, but in this section, we restrict our discussion to deployed research systems and commercial systems.

2.1 Non-MAS Solutions

Non-MAS TCSs are traditional traffic management systems utilized for traffic control purposes. Most conventional TCS in the US are still server-based. In this section, we focus on non-MAS cloud-based solutions. Only a very few Non-MAS TCSs are cloud-based.

2.1.1 Non-MAS, Cloud-Based. Solutions include Yunex Traffic [28, 42] and the widely deployed SCATS commercial system[26].

This work is licensed under a Creative Commons Attribution International 4.0 License.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

Yunex Traffic, a 2021-established Traffic Control System (TCS), optimizes traffic flow and enhances safety using SCOOT [31, 38] to process real-time sensor data for queue predictions, informing an optimizer for timing adjustments. Through its cloud-based *Yunex Symphony*, it offers real-time traffic monitoring interfaces. Despite some decentralized traits, the core decision-making via SCOOT is centralized, contrasting with the decentralized nature of multi-agent systems.

The Sydney Coordinated Adaptive Traffic System (SCATS) is a commercial Traffic Control System (TCS) developed in Sydney, Australia, that has been in use since the 1970s. SCATS operates on three levels of hierarchy. The Central Manager manages and configures the regional servers, and the Region processes TRAFF data and runs the SCATS algorithm. TRAFF processes the inputs and provides data to the SCATS core platform, which runs on servers hosted on the cloud. Despite being cloud-based, SCATS still faces the constraints of traditional centralized systems due to the centrality of signal timing decisions. Any issue with the Central Manager can adversely affect the traffic light networks, underscoring their limitations.

For research solutions, Zhang and Zhou [43] propose a TCS that uses a cloud computing platform to execute a bi-level optimization strategy for signal timing plans. The process is hierarchical, with *Coordinated Control* at the top level and *Distributed Controls* on the second level. The plans are then communicated to intersection controllers via wireless base stations. The research targeted optimizing traffic in Beijing, China but was never fully deployed to production. The solution's main limitations are the single points of failure at all levels in the hierarchy.

2.2 MAS Solutions

There are currently a very few deployed MAS solutions for traffic control. In multi-agent TCSs, the intersection controllers are responsible for defining and optimizing traffic signal timing plans in real time for their respective intersections, without higher-level control, supervision, or coordination.

2.2.1 MAS, Non-Cloud Based. The only two documented deployed MAS solutions are SURTAC[32] and *DALI*[33–36]. Both systems are deployed using a server-based architecture.

In SURTAC, the agents run as part of the controller software which executes using a conventional server-based architecture, whereas in *DALI* the agents are decoupled and run on remote servers.

SURTRAC was developed in 2013 by Smith et al. [29], and commercialized after the pilot deployment of SURTRAC in the East Liberty neighborhood of Pittsburgh, Pennsylvania. In SURTRAC, at each intersection, an advanced video camera sends video data to a video detection unit located in the intersection controller box. The software component, called SURTRAC Processor, also located in the intersection controller box, receives the processed data from the video detection unit and information about incoming vehicles from the direct intersection neighbors, and uses forward dynamic programming to calculate signal timing plans. Although efficient, SURTRAC's main limitations are the need for expensive hardware (e.g., advanced cameras, advanced detectors placed on the upstream end points of entry approaches) and the signal timing planning done in isolation at the intersection level with information from direct neighbor controllers.

DALI is a collaborative multi-agent TCS successfully deployed in 2019 in Richardson, a suburb of Dallas, Texas. In *DALI*, controller agents run in a server-based infrastructure and connect to intersection controllers via VPN. They continuously communicate and coordinate with each other through direct links. Unlike SURTRAC, the decision-making process for a signal-timing change is collaborative and involves the feedback of all controller agents affected by a change. The collaboratively-defined timing strategy not only improves the traffic flow at the intersection level but also does not create congestion at downstream intersections. In addition, unlike SURTRAC, *DALI* uses existing basic sensors (e.g., inductive loops) and does not require additional expensive hardware.

Although the multi-agent solutions discussed above offer greater scalability compared to non-MAS TCSs, they are deployed in non-cloud environments, which lack on-demand scalability. In addition, for TCSs with controller agents operating in a server-based setup, expensive high-speed direct connections are necessary to guarantee low latency.

2.2.2 MAS, Cloud-Based. Multi-agent TCSs in this category take the extra step of utilizing cloud-based technologies. Cloud-based solutions are categorized into *non-cloud-native* and *cloud-native.*

MAS, Non-Cloud-Native. Non-cloud-native solutions refer to software applications and technologies not specifically designed to run in a cloud computing environment, often relying on traditional, monolithic architectures where the application is developed as a single, indivisible unit but running in the cloud[11].

Although most attempts at TCS cloud implementation [12, 43] have been non-cloud native approaches, we are not aware of any non-cloud-native multi-agent TCS. Nevertheless, Li, Chen, & Wang [19] discuss how to setup and deploy a simulated multi-agent TCS in the cloud using virtual machines. This model adopts a 'lift and shift' approach, encapsulating the application's four distinct layers—application, platform, unified source, and fabric—and deploying them onto virtual machines within the cloud. While this method facilitates a straightforward migration to the cloud, it retains inherent limitations common to non-cloud solutions, such as restricted scalability and the presence of single points of failure.

MAS, Cloud Native System. Cloud-native solutions are applications and services specifically designed for the cloud's dynamic and scalable environment, making use of microservices architecture, containerization, and continuous integration/continuous deployment (CI/CD) practices [11, 23].

There are currently no MAS, cloud-native systems for traffic control. Nevertheless, Dahling et al. [8] discuss a cloud-native MAS platform called cloneMAP intended to facilitate the development of cloud-native MAS for IoT applications. Technologies such as Kubernetes, Docker, Cassandra, etcd, and GoLang are used to create a deployable cloud *Platform as a Service* (PaaS) to manage the software components. Although the proposed platform is geared toward cloud-based PaaS, the experiments were conducted as simulations executed on three physical machines, each running four virtual machines to execute the Kubernetes. No details are provided on how the solution was implemented in the cloud. In addition, cloneMap can only be deployed on Kubernetes. Other container orchestration systems such as Docker Swarm, Hasicorp Nomad, and Mesos cannot be used.

In this paper, we propose MATS-Cloud, a novel, generic architecture designed for the seamless deployment of multi-agent traffic control systems in the cloud. In addition to the architecture, our contribution involves leveraging cutting-edge cloud technologies, including cloud-managed serverless computing [6], Docker containers [9], and software-defined networking solutions[40]. To validate this architecture, we re-engineered and redeployed DALI, a multi-agent TCS, which originally operated on a conventional server-based architecture. With our solution, DALI, or any multi-agent TCS, can operate without disruption, scale up and down as needed, and maintain low latency.

3 SERVER-BASED AND CLOUD-NATIVE DESIGNS

The process of migrating a multi-agent traffic control system (MATS) into a cloud-native architecture involves various critical steps, including containerization of components and their dependencies, breaking down the monolithic TCS into microservices, orchestration, serverless computing, cloud-native data storage solutions, using a service mesh for networking, and utilizing monitoring and logging tools such as Datadog and Grafana to analyze and troubleshoot system issues.

In this section we begin by discussing a conventional server-based architecture used to implement and operate DALI. We then present MATS-Cloud, a cloud-native architecture designed for migrating any MATS where agents are decoupled from controllers, to the cloud.

3.1 Server-Based Architecture

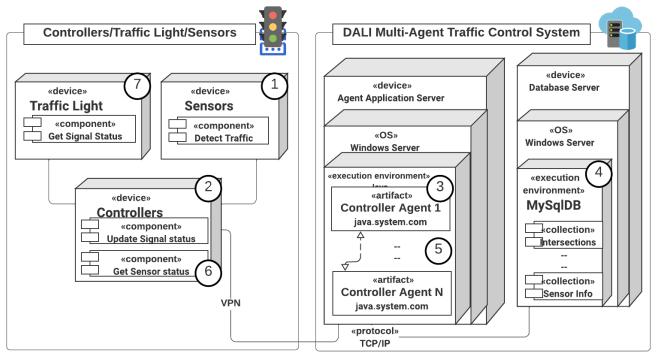

Figure 1: Deployment Diagram for the current server-based implementation of *DALI*

As shown in Figure 1, DALI is deployed on application servers running on a server-based data center. The controllers are located at the traffic intersections, and interactions between controllers and agents are managed via Virtual Private Network (VPN).

A conventional traffic control process begins with sensors detecting vehicles (Step 1 in Figure 1) and forwarding this data to the controller for processing (Step 2). Subsequently, the controller presents this data to the agent to formulate a signal timing plan (Step 3). To devise a signal timing plan, the agent integrates the data from the controller, along with historical data housed in the SQL database (Step 4). Moreover, due to the continuous exchange of information among the agents (Step 5), shared information such as detected vehicles and intersection statuses is factored into the formulation of the timing plans. Upon receipt, the controller executes the plan (Step 6) by sending a signal to the traffic lights via physical wires. This activates the appropriate circuits in the traffic lights, causing them to change color accordingly.

3.2 The MATS-Cloud Architecture

The decision to migrate a system to the cloud is primarily influenced by two factors: *latency* [16, 27] and *scalability* [10, 18].

Latency defined as the duration between a request and the receipt of a response, is a crucial performance measure for MATS, where controllers' agents cooperate to attain shared goals and coordinate their actions. From a cloud technology perspective, we use AWS microservices tools and the AWS Fargate serverless container solution to achieve low latency.

Scalability defined as the system's ability to handle an increasing number of requests without decrease in performance, ensures the system performance is maintained notwithstanding the workload. For a modern traffic control system, scalability on demand is particularly vital to dynamically accommodate increased system use demand and maintain the system's performance. Our proposed solution adopts a microservices architecture [14, 15], where a large application is divided into smaller independent parts, each with its domain of responsibility. Each service is designed to detect surges in usage and scale horizontally to accommodate requests.

Our proposed cloud-native architecture shown in Figure 2, uses a suite of cloud-native technologies, carefully selected to achieve the features mentioned above as well as *security*. Such a framework is necessary for a traffic management system that requires critical real-time data processing and dynamic decision-making. The three core components of the architecture include: *agents*, *communication*, and *databases*.

3.2.1 *Agents.*

Agent Core. At the heart of the system's intelligence are the DALI agents, written in the JAVA programming language. The agents are housed within Docker containers, which are standalone, executable packages of software that include everything needed to run an application: code, runtime, system tools, system libraries, and settings. The orchestration of these containers is managed through an ECS (Elastic Container Service) Cluster, a service provided by AWS that facilitates the running and management of Docker containers on a cluster of virtual servers. AWS Fargate, another component of this architecture, is a compute engine for Amazon ECS that eliminates the need to manage servers or clusters, thereby simplifying

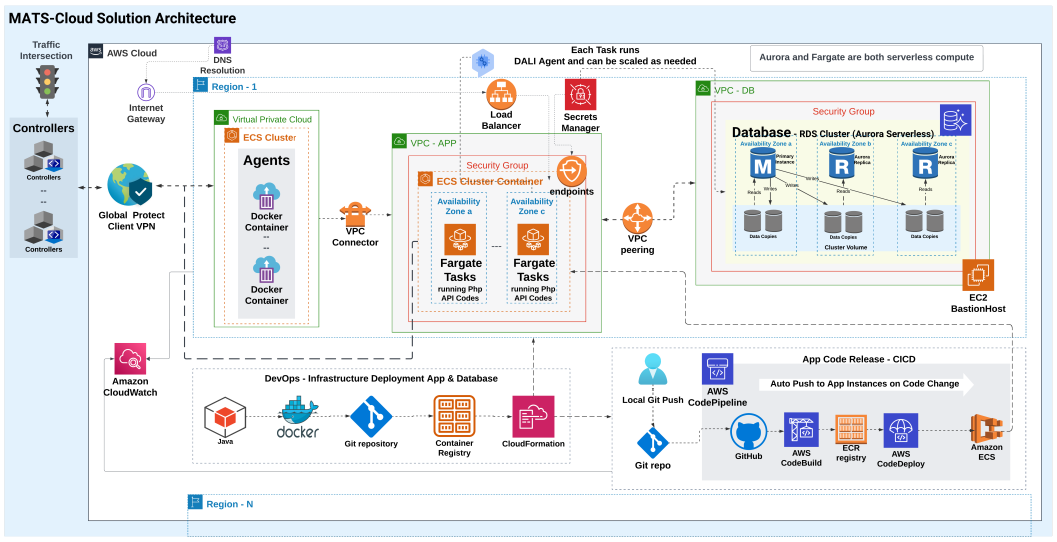

Figure 2: *MATS-Cloud* Solution Architecture - Single Region.

the process of running containers. By utilizing AWS Fargate, the complexities associated with server management, such as server provisioning or scaling, are removed.

MATS-Cloud Micro-Services. Microservices is an organizational approach where an application is divided into small, independent services, each focused on a single functional capability. At the core of MATS-Cloud microservices lies the Agent Services, each depicted as an individual microservice (see Figure 3). These autonomous, self-contained services manage various aspects of the agents' signal timing planning and traffic data analysis tasks. They operate within a security group that ensures secure access and functionality. The microservices interact with the DB cluster to facilitate efficient data management. Overall, this microservices configuration promotes high availability and low latency, ensuring optimized performance for real-time traffic management.

3.2.2 *Communications.*

Agent API. The agents' APIs are vital communication components within the system and are hosted on AWS Fargate (see Figure 3). Housed within Docker containers, the agents' APIs are managed by an Amazon ECS cluster situated within a specifically designated Virtual Private Cloud (VPC). This setup ensures that the agents' APIs can reliably process requests and perform tasks in a scalable and isolated environment. Load balancers distribute incoming network traffic across the various targets of Docker containers running the agents' APIs to ensure high availability and fault tolerance. Security groups act as virtual firewalls, regulating inbound and outbound network traffic to the agents' APIs, thereby enabling secure communication channels.

Agent-to-Agent Communications. These communications are facilitated by the agents' APIs. As mentioned above, to ensure a secure and isolated network environment, agents and their APIs are deployed in separate Virtual Private Clouds (VPCs). To enable communication between the agents, a VPC connector is required. This connector provides a handshake mechanism between the agents and their APIs.

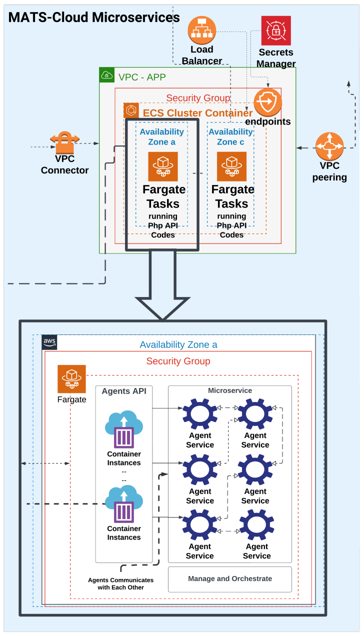

Figure 3: *MATS-Cloud* Microservices

AWS VPC provides a secure and isolated network for this communication. AWS VPC networking features, including load balancers and VPC peering, manage traffic routing between agents in different subnets or availability zones, while security groups serve as virtual firewalls to control network traffic and ensure secure communication.

Agent-to-Controller Communications. Communications between the agents and the traffic controllers at intersections is secured through a Global Protect Client VPN, with processing being executed by the Agent API.

3.2.3 ***Databases***. Our architecture employs two distinct types of databases to manage different data streams effectively:

(1) *Local Data Storage (Agent Knowledge-Base):* This database stores operational data crucial for immediate traffic decision-making and is designed for high availability and responsiveness to the agents' needs.
(2) *Collaboration Database Storage (Historical Information):* This database archives historical traffic data, which is used to inform long-term strategies and enhance system intelligence over time.

The adoption of AWS Aurora Serverless as our database solution is due to its automatic scalability in response to fluctuating demand, maintaining cost-efficiency and high performance across varying workloads.

In addition to the core components of a MATS cloud-native architecture, it is important to address concerns surrounding security and continuous integration and deployment. These aspects are briefly discussed in the following sections.

3.2.4 Security and Monitoring. The integration of AWS Secrets Manager ensures secure credential management, and AWS CloudWatch offers comprehensive monitoring and logging, both of which are vital for maintaining the system's security and performance.

AWS Secrets Manager and AWS CloudWatch are integral components within the MATS-Cloud solution for security and operational efficiency. Secrets Manager securely handles sensitive credentials necessary for system components to interact, mitigating risks associated with credential exposure. The database credentials are stored in Secrets Manager, and the Agents API interacts with the database using these credentials. CloudWatch provides real-time monitoring and logging capabilities, ensuring the system's health and facilitating quick responses to any operational issues, thus supporting overall system performance and stability.

3.2.5 Continuous Integration and Deployment (CI/CD). The CI/CD pipeline is an essential part of the MATS-Cloud solution, automating application updates for a seamless development process as shown in the Figure 2. Starting with code changes in a version control system such as GitHub, AWS CodeBuild compiles and tests the code to avoid errors. Successful builds are stored as Docker images in AWS ECR, ready for AWS CodeDeploy to release the application with minimal downtime. Integrated with AWS ECS, this pipeline ensures high-availability and facilitates the swift, secure deployment of updates, keeping the application current with the latest advancements and security standards.

4 IMPLEMENTATION AND EVALUATION

Experimental Results

Using the generic MATS architecture discussed in Section 3.2, we implemented and deployed DALI-Cloud. This section outlines the experimental setup used to evaluate the latency and scalability of DALI-Cloud compared its traditional server-based implementation. We utilized three observability tools: APIMetrics, DataDog, and Grafana, to measure various system metrics for each design option.

To assess the performance of our proposed cloud-native solution, denoted as *MATS-Cloud*, we conducted multiple experiments comparing the server-based and cloud native - serverless container alternatives.

These observability tools played a crucial role in providing insights into system behavior, identifying performance bottlenecks, and suggesting optimization opportunities. We selected APIMetrics, DataDog, and Grafana for their cloud-focused design, real-time monitoring, and broad metric tracking capabilities [17, 25].

(1) **APIMetrics** : A tool specialized in real-time API performance, reliability, and security tracking. Ideal for API-dependent organizations, offering performance testing, monitoring, and reporting.
(2) **DataDog** : An analytics and monitoring platform, providing real-time visibility into IT infrastructure, applications, and logs. Known for performance monitoring, log management, tracing, and alerting. Cost-effective, user-friendly, and supports multi-cloud configurations.
(3) **Grafana** : An open-source observability platform with comprehensive features for cloud and microservices monitoring. Includes end-to-end tracing, root cause analysis, real-user monitoring, and cloud/microservices monitoring. Focuses on modern technologies.

4.1 Data Collection and Monitoring:

For a comprehensive understanding of the performance metrics, these three monitoring solutions were utilized to compare the server-based implementation of DALI (referred to as Virtual Machine in this section) and DALI-Cloud (referred to as Fargate). The Synthetic Monitoring & Continuous Testing feature of the three tools (1, 2, & 3) facilitated data recording over a span of 14 days.

4.2 Experiment Setting

The objective of our experimentation was to understand the performance differences between the VM and Fargate deployment architectures. This section delineates the configurations and parameters used for both implementations and the methodology employed for data collection.

Application Deployment	**Database Deployment**
Platform: VM Instances: CPU - 2 cores Memory: 4096 MB Availability: 3 instances Networking: Within a Virtual Private Network Storage: Attached 30 GB volume to each instance Operating System: Windows Server 2022	Platform: VM Instances: CPU - 2 cores Memory: 4096 MB Networking: Within a Virtual Private Network Storage: Attached 30 GB volume to each instance Operating System: Windows Server 2022

Table 1: Server-Based - Virtual Machine (VM) Configuration

4.3 Latency

Our latency testing methodology involves load, stress, and component testing techniques. Load testing simulates high-traffic conditions, stress testing pushes system boundaries, and component testing assesses individual elements. By employing these approaches,

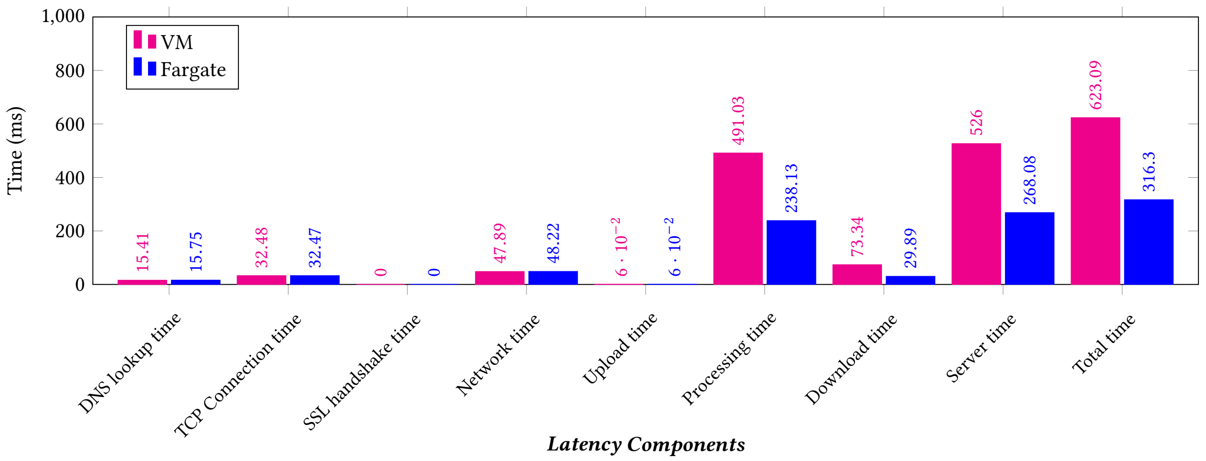

Figure 4: Average Latency Per Metric

Application Deployment	Database Deployment
Clusters: Three instances of type `t2.medium` Task Definition: - Memory: 4096 MB - CPU: 2048 units Container Definition: - CPU: 2048 units - Memory: 4096 MB Networking: Positioned within a VPC	Service: Amazon Aurora (compatible with MySQL 8) Networking: Located within the same VPC as the application Engine: Serverless ACUs (Aurora Capacity Units): Min. of 8 and a max. of 64

Table 2: Serverless Container-Based - Fargate Configuration

we aim to evaluate system performance under diverse operational circumstances, preemptively identify potential issues, and optimize system performance.

We used APIMetrics, Datadog, and Grafana for synthetic monitoring of application latency over a 14-day period. These tests covered various aspects of application latency, including authentication, traffic signal timing planning, and controller sensor data retrieval. Geographic diversity was considered, helping identify regional latency anomalies.

For all the evaluations discussed in the remainder of this section, the latency metrics collected included *dns lookup time*, *tcp connection time*, *ssl handshake time*, *network time*, *upload time*, *processing time*, *download time*, *server time*, and *total time*, measured in milliseconds.

4.3.1 *Evaluation of Average Latency per Metric.*

To evaluate the average latency per metric, we aggregated data from all the three observability tools and calculated the average time in milliseconds for each latency metric. The analysis of latency components between VM and Fargate, shown in Figure 4, revealed comparable values for *dns lookup time*, *tcp connection time*, and *network time* across both environments.

Notable disparities were observed in *processing time*, with VM exhibiting a higher duration, contributing to its overall elevated latency compared to Fargate. Additionally, VM's *download time* surpassed Fargate's, further widening the latency gap.

Fargate consistently showed lower total latency compared to VM, with the impact of *processing time* and *server time* being more pronounced in the VM environment.

4.3.2 *Evaluation of Latency using APIMetrics, DataDog and Grafana.*

Comparing latency metrics from APIMetrics, Datadog, and Grafana, instead of relying solely on one, is essential for a comprehensive understanding of system performance. It offers diverse insights through specialized monitoring capabilities, ensures the reliability of analysis findings through cross-tool validation, and identifies the strengths and weaknesses of each tool. Figure 5 shows the average latency comparison across the three tools.

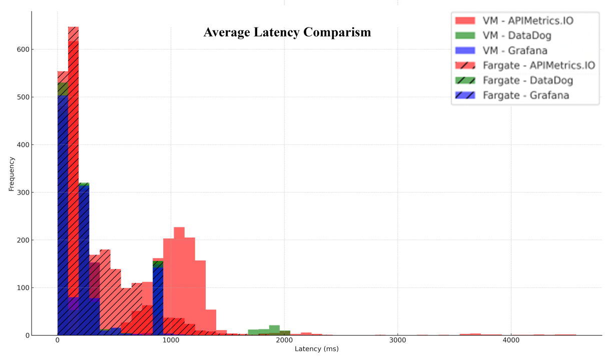

Figure 5: Average Latency Comparison for APIMetrics, DataDog, and Grafana

Central Tendencies. Fargate exhibits a consistent mean latency of approximately 259.74 ms across Grafana, APIMetrics, and DataDog datasets, compared to VM which has a mean latency of 399.78 ms. Additionally, Grafana displays a higher median latency for Fargate compared to APIMetrics and DataDog, signifying favorable performance consistency.

Tool-agnostic Consistency. Across different metrics tools, Fargate consistently outperforms VM in latency metrics, reinforcing the robustness of Fargate's performance metrics.

The histogram analysis provides a granular insight into VM and Fargate performance, endorsing Fargate's consistently favorable latency metrics. The alignment of observations among different tools positions Fargate as an efficient cloud service option.

4.3.3 ***Evaluation of Average Latency over Time.*** We conducted a 14-day latency analysis, focusing on maximum, minimum, and average latency metrics, to compare DALI-Cloud (Fargate) with the server-based solution (VM). We chose APIMetrics for the metrics data because of its capability to capture and log each metric component in a granular and detailed manner by seconds, making it suitable for analyzing average daily latency over time.

Figures 6, 7, and 8 show the trend for daily maximum, minimum, and average latency.

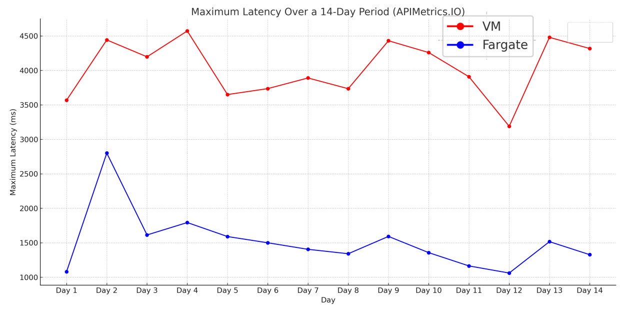

Figure 6: Maximum Latency Over a 14-Day Period

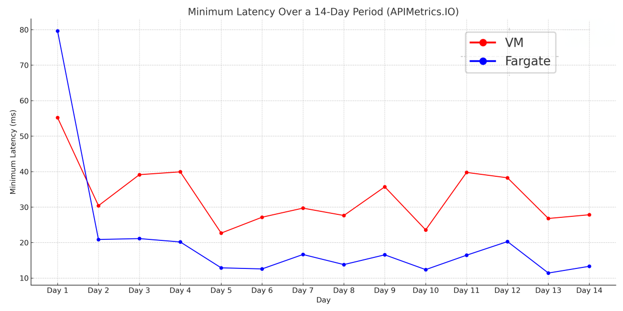

Figure 7: Minimum Latency Over a 14-Day Period

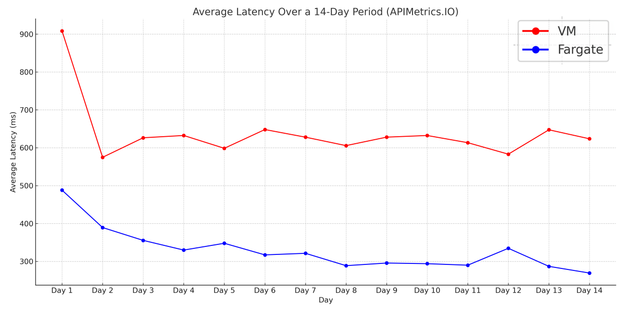

Figure 8: Average Latency Over a 14-Day Period

Maximum Latency. The analysis in Figure 6 reveals that VM consistently displays higher maximum latency when compared to Fargate. Notably, both VM and Fargate encounter latency spikes around Day 9, indicating a shared performance characteristic during this period, which could suggest that they encountered a common issue or demand spike that affected their performance.

Minimum Latency. As shown in Figure 7, throughout the analysis period, Fargate consistently maintains lower minimum latency. Despite the lower values in Fargate, both VM and Fargate exhibit stable minimum latency. This stability suggests reliable baseline performance from both environments when operating at their most efficient state, assuming identical configurations.

Average Latency. The trend in Figure 8 shows VM with a higher average latency than Fargate consistently. Although both VM and Fargate experience spikes in average latency, Fargate demonstrates a relatively stable average latency, suggesting a more steady performance in comparison to VM.

In summary, over the span of 14 days, a comparative analysis of VM and Fargate revealed notable performance distinctions. Fargate consistently outshone VM by exhibiting lower variability in latency metrics, thereby confirming its more stable and reliable performance. Unlike VM, which encountered latency spikes, Fargate maintained a steady latency, further emphasizing its superior performance. Additionally, Fargate held a distinctive advantage by maintaining a lower average latency and achieving the lowest latency metrics, which indicates its optimal performance.

4.4 Scalability

Our methodology for scalability testing encompasses load testing, stress testing, capacity testing, and horizontal scalability testing. Similar to latency testing, our strategy for scalability testing is comprehensive, integrating load and stress testing with the added dimension of capacity and horizontal scalability testing.

4.4.1 ***Evaluation of Scalability over Time.*** We conducted a 14-day scalability analysis focused on total time to compare the DALI-Cloud (Fargate) with the server-based solution (VM). We relied on metrics data from all the three tools. Figure 9 provide insights into scalability timing metrics across the three metrics tools.

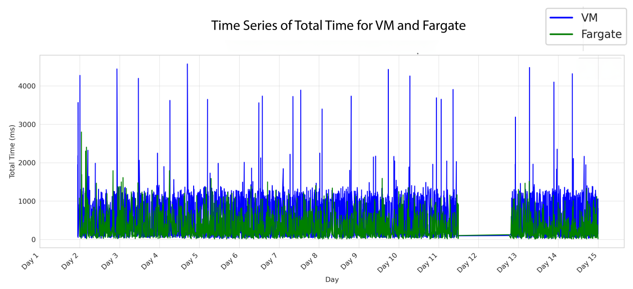

Figure 9: Time Series analysis of Total Time for APIMetrics, DataDog, and Grafana

The time series analysis showcases VM's response time fluctuations, possibly due to varying workloads or system inefficiencies, while Fargate exhibits consistent performance, hinting at resilience under diverse load conditions. This consistency positions Fargate as a preferable option for scalability.

4.4.2 ***Evaluation of Scalability using APIMetrics, DataDog and Grafana.*** Comparing scalability metrics across APIMetrics, Datadog, and Grafana is crucial for understanding system performance. It ensures analysis reliability through cross-tool validation and identifies each tool's strengths and weaknesses. Table 3 presents the mean, median, and max scalability comparison among these tools.

Scalability assesses a system's ability to handle increased loads without performance degradation. Latency, measured as request processing time, is a key indicator of scalability. Consistent low latency under heavy loads signifies good scalability, while latency spikes may indicate issues.

We compare scalability between VM and Fargate using response time data as shown in Table 3.

Tool	Env.	Mean (ms)	Median (ms)	Max (ms)
APIMetrics.IO	VM	771.59	690.67	4572.32
DataDog	VM	326.12	258.13	1990.65
Grafana	VM	324.76	257.27	1998.24
APIMetrics.IO	Fargate	489.59	434.18	3565.12
DataDog	Fargate	237.27	190.00	885.72
Grafana	Fargate	239.87	190.95	1193.26

Table 3: Metrics Tools Comparison

Central Tendencies. The analysis indicates a slight monitoring impact with Fargate's average response time from APIMetrics being 489.59 ms, marginally higher than that observed on DataDog and Grafana. On the other hand, VM's median response time, as noted via Grafana, demonstrates a steady performance, only slightly lagging behind APIMetrics while outperforming DataDog. Overall, the data from all tools reveal similar trends, suggesting that Fargate deployments offer better scalability than VM deployments.

Tool-specific Insights. Across different metrics tools, Fargate consistently surpasses VM in scalability metrics, displaying narrower and more focused response time distributions. The convergence of findings across these tools reinforces Fargate's superior scalability attributes.

With its consistent and favorable metrics affirmed by various tools, Fargate stands out as an efficient cloud service platform with respect to scalability, making it a viable choice for modern multi-agent TCS deployments.

4.4.3 ***Evaluation of Failure Rate over Time.*** To strengthen our argument in favor of the DALI-Cloud (Fargate) over the server-based (VM) approach, we analyze daily *failure rate* trends over 14 days. We focused on the metrics data from APIMetrics due to its ability to capture and log the failure rates in a granular and detailed manner.

As shown in Figure 10, Fargate excels by displaying lower rates, which underscores its reliability, especially under high loads. Conversely, VM showcases slightly elevated failure rates, signaling occasional challenges in request handling.

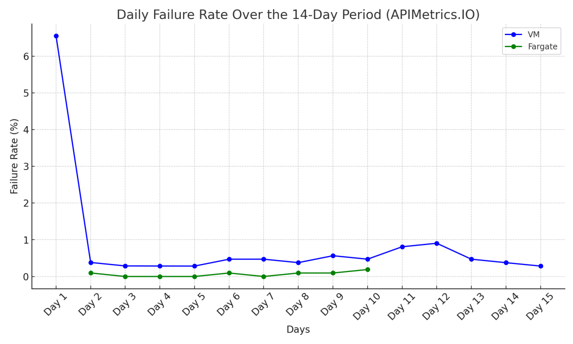

Figure 10: Daily Failure Rate Over a 14-Day Period

In summary, Fargate consistently outperforms VM in terms of scalability, evidenced by lower failure rates and higher daily request processing. Moreover, Fargate maintains stable server times across varying loads, indicating a level of processing proficiency. This proficiency extends to request handling where Fargate registers fewer errors, further consolidating its position as a more scalable solution in comparison to VM.

5 CONCLUSION

In this paper we presented MATS-Cloud, a generic architecture for the deployment of multi-agent traffic control systems (MATS) in the cloud, using AWS technologies. We implemented and deployed this architecture for DALI, a MATS currently operating using a server-based architecture. We utilized three observability tools, namely APIMetrics, DataDog, and Grafana, to evaluate DALI-Cloud and its server-based implementations. Experimental results highlighted the cloud implementations' effective latency mitigation for real-time traffic control and dynamic scalability ensuring responsiveness in high-demand scenarios.

Although the presented architecture was defined with AWS in mind, most components have their counterparts in other public cloud platforms. Our future work includes investigating an agnostic architecture that can accommodate Google Cloud Platform and Microsoft Azure, and exploring multi-cloud hybrid solutions.

ACKNOWLEDGMENTS

We thank the MAVS (Multi-Agent and Visualization Systems) research team at UT Dallas for their dedication to the success of the DALI project.

Our deepest appreciation goes to Dr. Behnam Torabi, whose guidance and expertise in DALI were invaluable in overcoming challenges associated with the design of MATS-Cloud.

We also express our gratitude to the City of Richardson, Texas, for their instrumental partnership and support, which played a significant role in our research and deployment efforts.

REFERENCES

[1] Sisay Tadesse Arzo, Zeinab Akhavan, Mona Esmaeili, Michael Devetsikiotis, and Fabrizio Granelli. 2022. Multi-agent-based traffic prediction and traffic classification for autonomic network management systems for future networks. *Future Internet* 14, 8 (2022), 230.

[2] Michael Balmer, Nurhan Cetin, Kai Nagel, and Bryan Raney. 2004. Towards truly agent-based traffic and mobility simulations. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.* IEEE, 60–67.

[3] Ana LC Bazzan and Franziska Klügl. 2014. A review on agent-based technology for traffic and transportation. *The Knowledge Engineering Review* 29, 3 (2014), 375–403.

[4] Saumya Bhatnagar, Francesco Percassi, Rongge Guo, Mauro Vallati, Lee McCluskey, and Keith McCabe. 2023. Automated Planning for Generating and Simulating Traffic Signal Strategies. In *32nd International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence.

[5] Jonathan J Buonocore, Ernani Choma, Aleyda H Villavicencio, John D Spengler, Dinah A Koehler, John S Evans, Jos Lelieveld, Piet Klop, and Ramon Sanchez-Pina. 2019. Metrics for the sustainable development goals: renewable energy and transportation. *Palgrave Communications* 5, 1 (2019).

[6] Gustavo André Setti Cassel, Vinicius Facco Rodrigues, Rodrigo da Rosa Righi, Marta Rosecler Bez, Andressa Cruz Nepomuceno, and Cristiano André da Costa. 2022. Serverless computing for Internet of Things: A systematic literature review. *Future Generation Computer Systems* 128 (2022), 299–316.

[7] Kuei-Hsiang Chao, Ren-Hao Lee, and Meng-Hui Wang. 2008. An intelligent traffic light control based on extension neural network. In *Knowledge-Based Intelligent Information and Engineering Systems: 12th International Conference, KES 2008, Zagreb, Croatia, September 3-5, 2008, Proceedings, Part I 12*. Springer, 17–24.

[8] Stefan Dähling, Lukas Razik, and Antonello Monti. 2021. Enabling scalable and fault-tolerant multi-agent systems by utilizing cloud-native computing. *Autonomous Agents and Multi-Agent Systems* 35 (2021), 10.

[9] Docker. 2022. Docker overview | Docker Documentation. http://tiny.cc/docker. (Accessed on 09/30/2023).

[10] dzone. 2022. Scalability and High Availability - DZone Refcardz. https://dzone.com/refcardz/scalability. (Accessed on 09/30/2023).

[11] Dennis Gannon, Roger Barga, and Neel Sundaresan. 2017. Cloud-native applications. *IEEE Cloud Computing* 4, 5 (2017), 16–21.

[12] Nihal Gaouar and Mohamed Lehsaini. 2021. A cloud computing based intelligent traffic control system for vehicular networks. In *Proceedings of the 4th International Conference on Networking, Information Systems & Security*. 1–5.

[13] Harsh Goel, Yifeng Zhang, Mehul Damani, and Guillaume Sartoretti. 2023. SocialLight: Distributed Cooperation Learning towards Network-Wide Traffic Signal Control. *arXiv preprint arXiv:2305.16145* (2023).

[14] Google. 2022. What Is Microservices Architecture?| Google Cloud. http://tiny.cc/micro-services. (Accessed on 09/30/2023).

[15] Rowan Haddad. 2022. Monolith vs Microservices Architecture: How to Migrate with Feature Flags. https://www.abtasty.com/blog/migrating-from-monolith-to-microservices//. (Accessed on 09/30/2023).

[16] Sujatha Krishanmoorthy, Zhangyu Wei, Yihuai Zhang, Hao Jin, and Hao Dong. 2020. A Study on Optimization of Network latency and Pocket loss Rate. In *IOP Conference Series: Materials Science and Engineering*, Vol. 937. IOP Publishing, 012054.

[17] Łukasz Kufel. 2016. Tools for distributed systems monitoring. *Foundations of computing and decision sciences* 41, 4 (2016), 237–260.

[18] Sebastian Lehrig, Hendrik Eikerling, and Steffen Becker. 2015. Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics. In *Proceedings of the 11th international ACM SIGSOFT conference on quality of software architectures*. 83–92.

[19] ZhenJiang Li, Cheng Chen, and Kai Wang. 2011. Cloud computing for agent-based urban transportation systems. *IEEE intelligent systems* 26, 1 (2011), 73–79.

[20] Junfan Lin, Yuying Zhu, Lingbo Liu, Yang Liu, Guanbin Li, and Liang Lin. 2023. DenseLight: Efficient Control for Large-scale Traffic Signals with Dense Feedback. *arXiv preprint arXiv:2306.07553* (2023).

[21] Yiling Liu, Guiyang Luo, Quan Yuan, Jinglin Li, Lei Jin, Bo Chen, and Rui Pan. 2023. GPLight: grouped multi-agent reinforcement learning for large-scale traffic signal control. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. 199–207.

[22] United Nations. [n.d.]. THE 17 GOALS | Sustainable Development. https://sdgs.un.org/goals. (Accessed on 10/06/2023).

[23] Newrelic. 2022. Cloud Native Is the New Normal: Is Your Environment Optimized for Success? https://newrelic.com/sites/default/files/2021-08/cloud-native-is-new-normal.pdf. (Accessed on 09/30/2023).

[24] Johannes Nguyen, Simon T Powers, Neil Urquhart, Thomas Farrenkopf, and Michael Guckert. 2021. An overview of agent-based traffic simulators. *Transportation research interdisciplinary perspectives* 12 (2021), 100486.

[25] Ayman Noor, Devki Nandan Jha, Karan Mitra, Prem Prakash Jayaraman, Arthur Souza, Rajiv Ranjan, and Schahram Dustdar. 2019. A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments. In *2019 IEEE 12th international conference on cloud computing (CLOUD)*. IEEE, 156–163.

[26] NSW. 2022. SCATS. https://www.scats.nsw.gov.au. (Accessed on 09/30/2023).

[27] Katia Obraczka and Fabio Silva. 2000. Network latency metrics for server proximity. In *Globecom'00-IEEE. Global Telecommunications Conference. Conference Record (Cat. No. 00CH37137)*, Vol. 1. IEEE, 421–427.

[28] Diego Salzillo-Arriaga, Martin Hartmann, and Felipe Gonçalves-Martins. 2022. Environmental Traffic Management: Technology Scenarios and Outlook. (June 2022).

[29] Stephen Smith, Gregory Barlow, Xiao-Feng Xie, and Zachary Rubinstein. 2013. Smart urban signal networks: Initial application of the surtrac adaptive traffic signal control system. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 23. 434–442.

[30] TRL Software. 2022. TRANSYT - TRL Software. https://tinyurl.com/ttransyt. (Accessed on 09/30/2023).

[31] Aleksandar Stevanovic, Cameron Kergaye, and Peter T Martin. 2009. Scoot and scats: A closer look into their operations. In *88th Annual Meeting of the Transportation Research Board. Washington DC.*

[32] Rapid Flowt Tech. 2022. Surtrac: Intelligent Traffic Signal Control System. https://miovision.com/surtrac. (Accessed on 09/30/2023).

[33] Behnam Torabi, Rym Z Wenkstern, and Robert Saylor. 2018. A self-adaptive collaborative multi-agent based traffic signal timing system. In *2018 IEEE International Smart Cities Conference (ISC2)*. IEEE, 1–8.

[34] Behnam Torabi, Rym Z Wenkstern, and Robert Saylor. 2020. A collaborative agent-based traffic signal system for highly dynamic traffic conditions. *Autonomous Agents and Multi-Agent Systems* 34, 1 (2020), 1–24.

[35] Behnam Torabi and Rym Zalila-Wenkstern. 2020. DALI: An Agent-Plug-In System to" Smartify" Conventional Traffic Control Systems. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 2120–2122.

[36] Behnam Torabi, Rym Zalila-Wenkstern, Robert Saylor, and Patrick Ryan. 2020. Deployment of a Plug-In Multi-Agent System for Traffic Signal Timing. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 1386–1394.

[37] Transmax. 2022. Traffic Services – Transmax. https://tinyurl.com/Transmaxt. (Accessed on 09/30/2023).

[38] trlsoftware. 2022. SCOOT® - TRL Software. https://tinyurl.com/tscoot. (Accessed on 09/30/2023).

[39] Phuriwat Worrawichaipat, Enrico H Gerding, Ioannis Kaparias, and Sarvapali Ramchurn. 2023. Multi-agent Signalless Intersection Management with Dynamic Platoon Formation. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. 1542–1550.

[40] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. 2014. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials* 17, 1 (2014), 27–51.

[41] Yutong Ye, Yingbo Zhou, Jiepin Ding, Ting Wang, Mingsong Chen, and Xiang Lian. 2023. InitLight: initial model generation for traffic signal control using adversarial inverse reinforcement learning. In *IJCAI*.

[42] Yunex. [n.d.]. Yunex Traffic - Uniting what's next in traffic. https://www.yunextraffic.com/. (Accessed on 10/09/2023).

[43] Yongnan Zhang and Yonghua Zhou. 2018. Distributed coordination control of traffic network flow using adaptive genetic algorithm based on cloud computing. *Journal of Network and Computer Applications* 119 (2018), 110–120.