DARTS: Distributed Architecture for Real-Time, Resilient and AI-Compressed Workflows

Ragini Gupta
University of Illinois at Urbana-Champaign
Illinois, USA
raginig2@illinois.edu

Bo Chen
University of Illinois at Urbana-Champaign
Illinois, USA
boc2@illinois.edu

Shengzhong Liu
University of Illinois at Urbana-Champaign
Illinois, USA
sl29@illinois.edu

Tianshi Wang
University of Illinois at Urbana-Champaign
Illinois, USA
tianshi3@illinois.edu

Sandeep Singh Sandha
University of California, Los Angeles
California, USA
ssandha@ucla.edu

Abel Souza
University of Massachusetts Amherst
Massachusetts, USA
asouza@cs.umass.edu

Klara Nahrstedt
University of Illinois at Urbana-Champaign
Illinois, USA
klara@illinois.edu

Tarek Abdelzaher
University of Illinois at Urbana-Champaign
Illinois, USA
zaher@illinois.edu

Mani Srivastava
University of California, Los Angeles
California, USA
mbs@ucla.edu

Prashant Shenoy
University of Massachusetts Amherst
Massachusetts, USA
shenoy@cs.umass.edu

Jeffrey A. Smith
DEVCOM Army Research Laboratory
Maryland, USA
jeffrey.a.smith1.civ@army.mil

Maggie Wigness
DEVCOM Army Research Laboratory
Maryland, USA
maggie.b.wigness.civ@army.mil

Niranjan Suri
DEVCOM Army Research Laboratory
Maryland, USA
niranjan.suri.civ@army.mil

ABSTRACT

IoT (Internet of Things) sensor devices are becoming ubiquitous in diverse smart environments, including smart homes, smart cities, smart laboratories, and others. To handle their IoT sensor data, distributed edge-cloud infrastructures are emerging to capture, distribute, and analyze them and deliver important services and utilities to different communities. However, there are several challenges for these IoT-edge-cloud infrastructures to provide efficient and effective services to users: (1) how to deliver real-time distributed services under diverse IoT devices, including cameras, meteorological and other sensors; (2) how to provide robustness and resilience of distributed services within the IoT-edge-cloud infrastructures to withstand failures or attacks; (3) how to handle AI workloads are in an efficient manner under constrained network conditions. To address these challenges, we present DARTS, which is composed of different IoT, edge, cloud services addressing application portability, real-time robust data transfer and AI-driven capabilities. We benchmark and evaluate these services to showcase the practical deployment of DARTS catering to application-specific constraints.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**; **Real-time system architecture**; • **Information systems** → **Information systems applications**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

IoT, distributed infrastructure, real-time data transfer, computational offloading, video compression, anomaly detection

ACM Reference Format:
Ragini Gupta, Bo Chen, Shengzhong Liu, Tianshi Wang, Sandeep Singh Sandha, Abel Souza, Klara Nahrstedt, Tarek Abdelzaher, Mani Srivastava, Prashant Shenoy, Jeffrey A. Smith, Maggie Wigness, and Niranjan Suri. 2022. DARTS: Distributed Architecture for Real-Time, Resilient and AI-Compressed Workflows . In *Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED '22), July 25, 2022, Salerno,*

ApPLIED '22, July 25, 2022, Salerno, Italy
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9280-8/22/07…$15.00
https://doi.org/10.1145/3524053.3542742

Italy. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3524053.3542742

1 INTRODUCTION

The rapid evolution of Internet-of-Things (IoT) has transformed most Smart City applications to be distributed in nature with large data streams from connected sensors, digital machines, communication devices and users. The data generated from the networked subsystems require efficient management, real-time analytics and processing for applications across diverse IoT driven communities; smart transportation, smart buildings, smart grid, first responder systems and military operations, to name a few. With the increasing demand and growth of devices, most IoT applications require interconnectivity among billions of heterogeneous services, which makes scalability an important concern for the support of ultra large numbers of IoT services [10; 22].

Today, there are various wired and wireless communication protocols to support ubiquitous connectivity within disparate IoT services such as Wi-Fi, LTE, Bluetooth, Ethernet, etc. However, interoperability among these subsystems still remains a challenge due to the lack of global standardization in a large networked system of systems, which results in siloed applications. The multiple dimensions of scalability such as horizontal scalability to add or remove IoT nodes [5; 29] and vertical scalability to increase or decrease access to computational resources for a given IoT node [2; 6; 23] have been well-studied in the literature. However, scalability in terms of the wide-scale connectivity among IoT services and applications is still under explored. Additionally, since most IoT applications are deployed in resource constrained environments, existing architectures are strictly customized for specific IoT use-cases [1; 9; 12; 13; 25–27], which limits their re-usability and adaptability to diverse applications.

In a typical IoT architecture, IoT devices like communication devices and sensors collect and communicate data to a centralized IoT cloud, which processes the data to render services for different consumer applications. These sensors are required to ensure data integrity along with equally reliable transfer of the data to the cloud. At the communication level of the IoT paradigm, message oriented event streaming protocols such as Kafka, MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocols), XMPP (Extensible Messaging and Presence Protocol), AMQP (Advanced Message Queuing Protocol) and the classic HTTP [7; 20] are utilized for IoT applications to provide large scale connectivity. These protocols are lightweight and are well-suited for latency critical applications and for resource constrained environments of IoT with limited network bandwidth. However, the interoperability of these protocols for specific application requirements in IoT is under explored.

Similarly, technologies such as containers (e.g. Docker) are prevailing today as solutions to provide better portability and scalability for software deployment. These technologies can be leveraged in distributed IoT services to reduce the load of IoT deployment and ensure better reliability. Moreover, machine learning centric approaches have been well studied in the existing literature [3; 16], which can be utilized for efficient communication in IoT paradigms to reduce a significant amount of transferred data in unreliable and constrained network scenarios. While these technologies are well-explored in isolation, a unified platform that integrates these loosely coupled components is yet to be designed to make a somewhat decentralized architecture for IoT that is not limited by specific constraints of a single application. Due to these requirements, in this paper we propose a new distributed IoT architecture, called DARTS, designed for real-time, resilient and AI compressed workflows.

DARTS is an IoT edge-cloud infrastructure consisting of different services: (i) Cross-domain application deployment service (portability), (ii) Distributed data transfer service for client IoT data streams, and (iii) AI-Compressed workflow services (computational offloading, video compression, and data integrity). We analyze different alternative design choices for each of these services and present a learning experience in designing an end-to-end IoT infrastructure. We experiment and benchmark each service showing the expected application overheads, message transfer delays, and throughput achieved under different scenarios. Through DARTS, our goal is to highlight the challenges in developing scalable IoT infrastructure along with different design choices to shape the next generation of IoT applications.

The contributions of this paper are as follows:

- We propose, in detail, the requirements of a distributed robust infrastructure for IoT applications.
- To satisfy the requirements, we present DARTS, an architecture consisting of different services that address individual requirements. We study in detail each service with alternative choices.
- We benchmark the introduced services to highlight their pros and cons in real deployments.

The rest of the paper is organized as follows: Section 2 provides an overview of different requirements for the infrastructure including the data computation model and the distributed model. Section 3 introduces the DARTS architecture with it's integral components and services. Experimental setup, datasets used in evaluation and evaluations results are discussed in Section 4. Section 5 presents the conclusion.

2 REQUIREMENTS

An IoT network refers to network-enabled sensors on devices such as autonomous cars, weather monitoring systems, smart appliances, wearable and surveillance equipment, that communicate with other devices without the need for human intervention. Due to heterogeneity of devices and their sensing modalities, applications running over these infrastructures have several requirements. Given its distributed nature, high scalability, real-time processing (guaranteed response times), reliability, and zero downtime are the representative characteristics for heterogeneous applications in dynamic smart city environments. This can be facilitated with the help of decentralized sense-compute-action infrastructure that provides end-to-end uninterrupted services, ubiquitous network connectivity among different subsystems, and highly available services.

2.1 Data Computation Model

In a typical smart city deployment, heterogeneous sensing modalities on distributed sensors are collaboratively utilized to perceive the physical surroundings, spanning from 1D time-series sensing data (e.g., temperature, humidity, seismic, acoustic data), 2D image data, to 3D videos. At more complex environments, an actual deployment will have many of such sensors, with 2D and/or 3D grids of data (i.e., tensors). The information contained in different sensing modalities are known to complement each other, thus achieving more comprehensive understanding of the physical state. To simultaneously meet the efficacy, efficiency, and resilience requirements, the following key problems need to be addressed.

- **Information Fusion:** To maximally utilize the complementary information from heterogeneous sensing modalities, effective sensor data fusion needs to be performed [14; 21]. Considering the AI workload and distributed sensing nature, the fusion can happen at different processing levels (e.g., input data, latent feature, decision), and at multiple places (e.g., sensor, edge server, end user). Further, the fusion mechanism needs to be resilient such that a reasonable decision can still be made when only part of the sensed data are available.
- **Computation Scheduling:** Scheduling decisions should be made about when and where to process each type of sensor data to optimize the efficacy-efficiency tradeoff. The data can either be processed locally at the sensor, offloaded to the edge/cloud server, or processed in a distributed manner between IoT-edge-cloud. There exists a tradeoff between the computation and communication cost.
- **Data Transmission:** For sensor data that needs to be transmitted to the edge/cloud, effective data compression techniques (e.g., video codecs) should be applied to save the bandwidth consumption, and corresponding adaptation algorithms should be deployed to handle the network dynamics, such that the response efficiency on the sensor data is optimized under limited and dynamic network connections. When using the AI models as the end data consumer, the optimization objective can be different for the compression and adaptation algorithms, compared to conventional use cases where humans play the role of the data consumer.

2.2 Distributed Model

Real-time and resilient data distribution from a large number of heterogeneous sensors to consumer applications demands careful adoption of underlying services. Decoupling sensors and consumer applications is the key for scalability. To that end, publish-subscribe communication protocols represent an ideal choice.

As a special network, smart city networks are widely distributed across different industries. As mentioned, these networks are used to sense, analyze and take decisions in real-time with strict performance and data constraints. Due to these requirements, a hierarchical network topology can be used to securely connect sensing devices to edge and larger cloud sites hosting databases, machine learning (ML) models, and dashboards for data visualization and discovery.

Modern distributed applications no longer use monolithic architectures that scale only by replicating the whole application on multiple servers. Given the geographical distribution, developers prefer to architect software as a set of microservices that separate the functionality of the application into different services, like caching and database services. As an example, we can mention Distributed Inference and Machine Learning. These workloads can be modeled as microservices composed of multiple stages that can be split in different microservices and be deployed over multiple infrastructural layers. For instance, training can happen in the cloud where more resources are available, and inference (predictions) can execute at the edge due to lower latency. That is even more important in distributed IoT platforms, because of the dynamicity on the resources available to these applications. To tackle these problems, we present an architecture to improve and perform tests and experiments in a distributed IoT network testbed.

3 DARTS

DARTS consists of multiple services to enable distributed IoT infrastructure linking hundreds of sensing devices across multiple geographical locations to applications. The services in DARTS are (i) cross-domain application deployment, (ii) distributed real-time data transfer, and (iii) AI-compressed application workflows constitute the integral part of the DARTS infrastructure. These services are designed and implemented to meet the IoT requirements discussed in Section 2. Next, we elaborate on each of these services in detail.

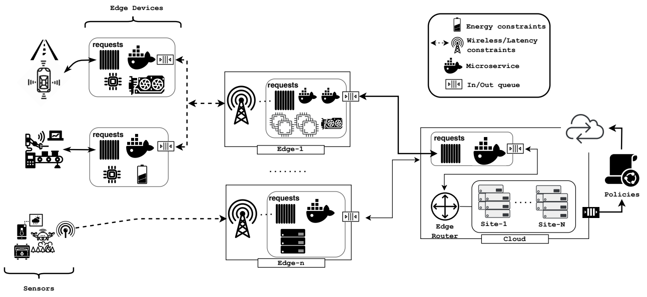


Figure 1: Cross-domain deployment of applications in DARTS

3.1 Cross-domain Application Deployment Service (Portability)

Figure 1 presents our tiered distributed IoT architecture. It can enhance distributed applications by retrieving more powerful cloud resources nearer to resource-constrained low-end devices at the edge, and by offloading compute tasks from end devices to the edge. Although not all applications can directly benefit from this architecture, the general and main end result is lower latency and higher bandwidth to client applications and end users that are directly interested in faster solution times. The architectural topology follows an hierarchy structure, depicting cloud sites with larger processing power to the right, and edge sites with low to medium processing capabilities in the middle (e.g., Edge-1). On the far left, there are the sensors and the edge devices, possessing lower – but local – computing capacities and some hard physical constraints (e.g., battery powered devices and low bandwidth intermittent network connections). Microservices located in all layers can communicate with one another, besides having the capabilities to offload processing to nearby edge or cloud sites, and onload when latency is important.

This topology is ideal for orchestrators like Kubernetes that use container systems such as Docker to run applications on behalf of users. Containers provide a lightweight method of packaging and deploying applications uniformly and consistently across different types of infrastructures because scaling from local to distributed setting is straightforward. This isolation mechanism used by systems such as Kubernetes leverage Linux kernel features, e.g., namespaces and cgroups, and at the same time they provide a uniform view of the underlying infrastructure. However, this infrastructure also comes with its own challenges, such as state management, placement, and performance variability.

Figure 1 shows different applications, where each has different needs – home, cars, wearable, weather, etc. – and different geographical locations. Thus, portability in processing and data communication/exchange are very important characteristics. With containerization, applications and algorithms are enhanced to work as services, enabling portability and configuration of complex and distributed workflows. This portability is important in terms of the development process and infrastructural compatibility. It also offers many notable benefits, like fault isolation, ease of management and security.

3.2 Distributed Real-time Data Transfer Service

To enable data transfer from hundreds of sensors across multiple geographical locations in DARTS, we adopt a publish-subscribe (Pub-sub) communication pattern. Pub-sub pattern decouples the producers (sensors) and consumers (applications), making them agnostic, which is key for scalability in diverse smart environments. Many different pub-sub protocols are proposed by researchers, such as Apache Kafka [11] and MQTT [4], which are widely used for real-time data distribution services among various endpoints, wherein the consumers are connected to publishers via brokers. These protocols primarily structure messages through communication channels called topics. However, it is challenging to select a particular protocol since each of them explicitly handles different use cases. MQTT is built for IoT use cases with many sensors, lightweight constrained devices, low delays, and high scalability. MQTT is typically used for streaming data from edge devices in a constrained network of tens of thousands of IoT clients. Apache Kafka provides long-term storage and buffering of messages, reprocessing of messages, and distributed brokers to provide high availability. Apache Kafka is used for data persistence in order to build highly scalable data pipelines and data integration across multiple subsystems and applications. Thus, there is no one-size-fits-all pub-sub protocol for distributed message delivery in a connected network of Smart-X (X: city, home, transportation) applications.

In DARTS, we adopt a heterogeneous pub-sub architecture built over MQTT and Apache Kafka, as shown in Fig 2. Fig 2 shows different smart city meteorological sensors (such as atmospheric temperature, humidity, pressure, and wind speed) sensing a region of interest, communicating sensor data via unified MQTT-Kafka API to the cloud applications using time-series database InfluxDB [19], and and web-based visualization dashboard Grafana [8] to store and monitor data respectively. Our goal via heterogeneous pub-sub architecture is to provide the best of MQTT and Apache Kafka based on specific use-case scenarios and to provide data communication even in the case of a single protocol failure. DARTS provides a unified API that allows users to select either MQTT or Apache Kafka for real-time data transfer. For use-cases where persistent data storage is preferred, developers can select Apache Kafka, whereas, for other workflows, MQTT is an ideal choice. Further, for Apache Kafka, we adopt distributed brokers across different geographical sites. In case a single protocol failure occurs, DARTS API uses the available working protocol. Distributed Apache Kafka can internally handle broker failures automatically. This resilience to failures is a critical requirement since computing nodes or a network partition can compromise communication among different endpoints of the system, making the end applications unresponsive.

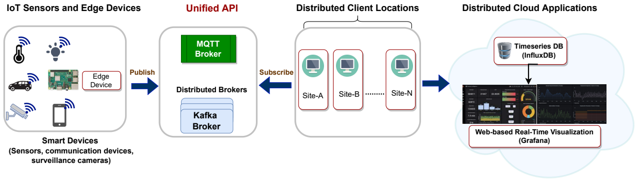

IoT Sensors and Edge Devices
Edge Device
Smart Devices
(Sensors, communication devices, surveillance cameras)
Publish
Unified API
MQTT Broker
Distributed Brokers
Kafka Broker
Subscribe
Distributed Client Locations
Site-A
Site-B
Site-N
Distributed Cloud Applications
Timeseries DB (InfluxDB)
Web-based Real-Time Visualization (Grafana)

Figure 2: DARTS with MQTT-Kafka integration

3.3 AI-Compressed Workflow Services

DARTS provides AI-Compressed workflow services that offload the computation at run-time based on available resources, reduce the amount of data transferred in constrained networks and enable data integrity. Next, we discuss each of these capabilities in detail.

3.3.1 *Computational offloading.*

This service is responsible for offloading part of the computation from the IoT devices to the edge server for better efficiency. To achieve this in DARTS, we adopt the compressive offloading technique proposed in [31], which, in general, is an asymmetric autoencoder network for data/intermediate feature compression. It consists of two parts: (1) A lightweight encoder that runs fast on the IoT device to convert the input data into latent feature maps with reduced spatial dimensions. (2) A heavyweight decoder that runs on the edge server to reconstruct the original input by gradually expanding the dimension of the encoded feature map. The encoding latency on the IoT device is traded for saving data transmission delay after compression. Compressive offloading integrates compressive sensing theory into the neural network design, which greatly increases the compression ratio while incurring negligible downstream DNN accuracy degradation.

Specifically, the proposed DARTS infrastructure focuses on compression services for image compression and computational offloading. As shown in Fig. 3, when an IoT device (e.g., the client) captures an image, it runs the compressive offloading encoder to compress the image into a feature map with smaller size. Then the feature map is transmitted through the network to a more powerful edge server equipped with a GPU. The edge server runs the compressive offloading decoder and reconstructs the compressed feature map back to the image. Finally, the downstream neural network (e.g. an object detection neural network) consumes the reconstructed image and finishes the application task.

3.3.2 *Video Compression.*

In recent years, end-to-end learned video codecs have been getting more attention by outperforming traditional video codecs in terms of coding efficiency. Nevertheless, the

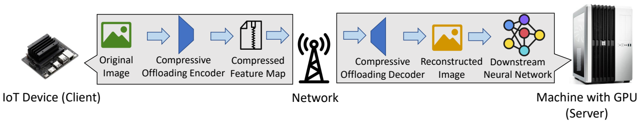

Original Image
Compressive Offloading Encoder
Compressed Feature Map
Compressive Offloading Decoder
Reconstructed Image
Downstream Neural Network
IoT Device (Client)
Network
Machine with GPU (Server)

Figure 3: The compression service to compress data on the client side and decompress data on the server side

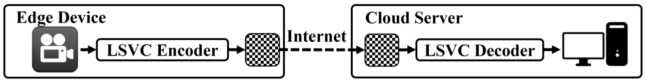


Figure 4: The video compression service, LSVC, compresses video streams on the edge device with the LSVC encoder and decompresses video streams on the cloud server with the LSVC decoder

encoding speed of existing learned video codecs is far from supporting live streaming due to the slow forward propagation problem.

In DARTS, a compression service, called LSVC, is implemented that encodes live video sources with a speed that supports live streaming and guarantees state-of-the-art coding efficiency. As shown in Figure 4, a live video streaming session involves an edge device, where the LSVC encoder compresses the raw video into a bitstream, and a cloud server, where the LSVC decoder decompresses the bitstream and the reconstructed video is presented to the human viewer.

LSVC slices a video into multiple group of pictures (GOP). In each GOP, there are an I-frame compressed by the image codec, Better Portable Graphics (BPG), and multiple P-frames compressed by a deep neural network. The compression network is based on a novel tree-based compression scheme, which allows parallelization in compression and trading GPU utilization for encoding speed. LSVC improves the coding efficiency with a space-time-aware entropy module that fully exploits spatiotemporal features during video compression.

3.3.3 Visualization and Anomaly Detection. DARTS extends the real-time sensor data monitoring and visualization capabilities as a service for the end users. In order to implement visualization and monitoring services, Grafana framework is used for real-time sensor data visualization in the cloud, as shown in Fig. 2. This web-based visualization platform allows the clients to gain meaningful insights from real-time sensor data in the cloud [9]. Grafana is connected to a time-series database in the cloud, InfluxDB, that stores multi-modal, time-stamped sensor data. In addition to sensor data visualization, we leverage monitoring capabilities for the users by providing anomaly detection services in the cloud. It is worth mentioning that weather applications rely on sensors that are deployed in rural areas under harsh atmospheric conditions which makes them highly prone to breakage and malfunctioning. This may cause erroneous measurements at the data collection site or data corruption during the transmission process, which can eventually misinform the consumer applications. In order to ensure sensor data integrity and data quality, anomaly detection methods using unsupervised machine learning is applied to proactively identify irregularities and deviations in the real-time sensor data. Due to the lack of ground truth labels for meteorological sensors, we apply Isolation Forest Method [17], an unsupervised machine learning algorithm to filter out anomalies from the sensor data.

4 EXPERIMENTAL EVALUATION

In this section, we elaborate on the experimental designs, datasets, and metrics used for multiple application scenarios to evaluate the proposed services of DARTS. For distributed real-time data transfer services, a cluster of distributed broker nodes with increasing number of producer-consumer applications is implemented to take measurements for network latency and throughput. For visualization and anomaly detection service, we demonstrate the performance of the anomaly detection method under different levels of anomaly distribution in the time-series sensor data. For computational offloading services, the proposed scheme is compared with existing compression algorithms with respect to evaluation metrics such as encoding-decoding time, network transmission latency, and end-to-end latency. For video compression services, the presented algorithm is evaluated with various baseline approaches in Wifi and Wifi (lossy) conditions, to compute the performance with respect to the rebuffering rates.

Finally, these application services were containerized using Docker (20.10) containers, so portability among the edge sites is assured. By using containers, we guarantee consistency and isolated environment for all workloads, besides repeatability and automation while running the experiments. Given the nature of some of these applications, containers allow for easier collaboration, modularity, and scalability. With such features, applications are ready to be deployed in distributed orchestrators such as Kubernetes. This characteristic is key when running heterogeneous application services and limited number of resources are available.

4.1 Experimental Setup

Distributed Real-time Data Transfer Services. We benchmark the performance of heterogeneous pub-sub architecture by simulating parallel producers and consumers, which applies to the scenario of having multiple sensors and processing applications. For Apache Kafka, we used v2.14, and for MQTT, we used Mosquitto message broker v1.3. To measure performance for a single broker, we simulate producers and consumers on a Intel Core i9-9820X CPU @ 3.30GHz server machine having 20 cores running Linux.

Video Compression. We validate the effectiveness of our approach by pushing video streams to the cloud server from the edge device over wireless networks. The edge device runs the video encoder on a Linux laptop with an Intel Core i9-8950HK CPU @ 2.90GHz and one NVIDIA GeForce GTX 1080 GPU. The cloud server runs the video decoder on a Linux desktop equipped with an Intel Core i7-9700K CPU @ 3.60GHz and two NVIDIA GeForce RTX 2080 Ti GPU. We use TCP to transmit the bitstream generated by different codecs naively. The frame rate of the video stream is configured to 30 fps. The edge and the cloud are placed inside two campus buildings. The edge device is connected to the network through WiFi while the client is connected via a 1Gbps Ethernet cable.

Computational Offloading. We deploy the encoder part of the compressive offloading framework on a Raspbery Pi 4 and the decoder part on a desktop with an Intel i9-9960X CPU 3.10GHz and a Nvidia GTX 2080 Ti GPU. We conduct the experiments under the network between the MSA testbed (see details in Sec. 4.2.1) and UIUC (around 1,000 miles distance).

4.2 Datasets

4.2.1 Sensor Data for visualization. Meteorological data sets were provided by the Atmospheric Science Center through the U.S. Army Combat Capabilities Development Command Army Research Laboratory Multipurpose Sensing Area (MSA), with cooperation from the USDA- Agricultural Research Service Jornada Experimental Range. Real time sensor data is collected from the MSA testbed real-time visualization, monitoring and anomaly detection service. The dataset consists of time-stamped weather parameters collected for a period of 1 month from sensors such as temperature, humidity, barometric pressure, sonic anemometer, soil temperature and solar irradiance sensor. These sensors are deployed at multiple locations within the New Mexico region. In the scope of this work, we limit our study to a subset of the collected sensor data i.e., atmospheric temperature and humidity. The sampling frequency of these atmospheric sensors is 1 Hz.

4.2.2 Image for vehicle recognition. We place 3 cameras across a field in a state park near UIUC. We also deploy 5 Raspberry Shakes which are equipped with seismic sensors in that field and collect the sensor recordings. The sensor recordings are processed by the Raspberry Shakes to perform vehicle recognition. A positive detection triggers the nearby camera to take a photo and then offload the photo to the edge server (a laptop) which runs YOLOv5 for image-based vehicle recognition. We collect data of 3 different types of vehicles, including a compact sedan, a sports car, and a mid size SUV.

4.2.3 Video for live streaming. We evaluate the presented video compression approach on the UVG dataset [18] containing 16 versatile 4K test video sequences captured at 50/120 fps.

4.3 Metrics

Distributed Real-time Data Transfer. The objective of the performance evaluation for distributed streaming was to compare the performance of the two IoT protocols, i.e., MQTT and Kafka with respect to metrics such as network throughput and end-to-end latency (in ms) for sending and receiving messages between the publishers-subscribers. Network throughput, on the other hand, is computed as the ratio of the total number of messages communicated per unit period of time among the distributed clients.

Video Compression. In video compression, we consider metrics of video quality measured by Peak-Signal-to-Noise Ratio (PSNR), encoding throughput calculated as the ratio of the bits per pixel used in compression to the frame rate of streaming, rebuffering ratio and frame rate.

Computational offloading. We evaluate the end-to-end latency as the metric, which includes the computation latency on the client side, network transmission latency and the computation latency on the server side.

4.4 Results

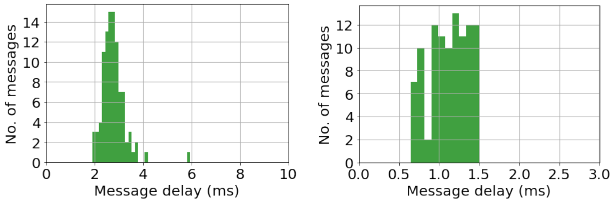
Distributed Real-time Data Transfer Service. We first benchmark the delay in messages as they transfer through the pub-sub system. The results for transferring 100 messages (of size 1 MB each) for a single publisher-subscriber are shown in Fig 5. These delays are measured on a machine discussed in Section 5.4. The publisher and subscriber clients are running locally on the server machine itself. Hence, these delays do not measure the network variations and allow users to directly compare Apache Kafka and MQTT protocols in the same settings. Average Apache Kafka delays (∼3ms) are higher than the average MQTT delays (∼1.2 ms). This is because Apache Kafka stores each message persistently on the disk as well, which contributes to the extra delay.

To benchmark the delays in distributed Apache Kafka brokers, we set up two brokers running on separate machines. The round-trip delay in publishing and receiving a message from a particular topic is dependent on the network delays between the publisher, subscriber clients, and the broker elected as the leader to store the topic partitions. To have a topic stored across brokers, we create a distributed topic. We measure delays for both wired and Wi-Fi connectivity between brokers. The results are shown in Fig. 6. As seen, the average delays are smaller with LAN connectivity, and with Wi-Fi connectivity, the delay spread has a long tail. These delays are very much dependent on the network characteristics. The distributed fault tolerance pub-sub system such as Apache Kafka comes at an extra cost for the network delays between the leader broker and the clients.

Next, we benchmark the throughput variation on both MQTT and Apache Kafka as the number of publisher-subscribers is varied. We increase the number of publishers and subscribers from 1 to 100 and measure throughput as the number of messages transferred by each publisher per second. In these experiments, we use a single broker for Apache Kafka, and the publisher and subscriber clients are running locally on the server machine itself. The results are shown in Fig. 7. As the number of publishers and subscribers increases, the average throughput per publisher decreases. This happens as the broker resources are throttled with increasing messages.

Further, we see that MQTT has a higher throughput initially, as the messages are not permanently stored as done by Apache Kafka. However, as the number of publisher-subscriber increased beyond 50, we see similar throughput in both protocols. These experiments show that the persistence storage of messages in Apache Kafka comes with a cost of more delays and less throughput in comparison to MQTT. Thus, our heterogeneous pub-sub architecture allows users to select their ideal choice for different use-cases. To handle protocol failures, we depend on the availability of another protocol. We adopt a timeout based on expected network delay and switch to a different protocol if message transfer does not succeed using the current protocol.

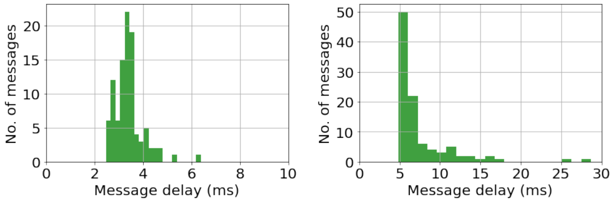
Visualization and Anomaly Detection Services. Real-time visualization application is implemented with Grafana for environmental monitoring using atmospheric sensors. Grafana provides web-based monitoring capabilities as shown in Fig. 8, which illustrates the real-time weather telemetry for temperature and humidity sensors collected over a span of few hours on one day. We leverage these visualization capabilities by integrating real-time anomaly detection application for the clients. Isolation Forest [17] algorithm is used for detecting anomalies by filtering out anomaly samples that are characterized as rare and different in the collected time-series data. Since the variability in real-time humidity and temperature sensor data is limited, we generate synthetic deviations in



(a) Apache Kafka Delays

(b) MQTT Delays

Figure 5: Apache Kafka and MQTT delays for a single broker



(a) Apache Kafka Delays on LAN

(b) Apache Kafka Delays on Wi-Fi

Figure 6: Apache Kafka delays for two distributed brokers

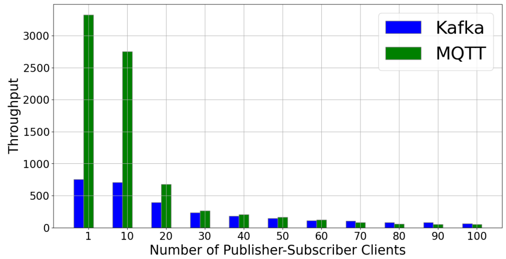


Figure 7: Throughput vs. number of publisher-subscriber clients for Apache Kafka and MQTT

the dataset to study anomalies emerging from sensor malfunction, incorrect operation or failures in data collection pipelines. These synthetic anomalies are injected in 10% of the sensor data using Gaussian noise distribution, with varying levels of noise parameters for mean (μ) and standard deviation (σ) as illustrated in Fig. 9. Fig. 10 demonstrates the accuracy of the Isolation Forest algorithm for different values of noise mean and standard deviation. The results indicate that the algorithm performs well with an accuracy of above 85% for noise distribution with higher standard deviation (and noise spread), that is, for noise mean and standard deviation values above 1.5. Moreover, the accuracy of the algorithm converges at 91% for the noise mean and standard deviation values above 2.7. This indicates that the algorithm is efficient for identifying anomalies that exhibit large amount of variation in the time-series data, and after reaching a convergence point the algorithm becomes robust to higher levels of anomaly distribution in the dataset.

Computational offloading. We compare compressive offloading, for computational offloading, to JPEG, which is a commonly used lossy image compression codec. We profiled the time cost of each step as well as the compressed image size and provide results in Table 1. The results show that compressive offloading can save the encoding and decoding time cost compared with JPEG. As

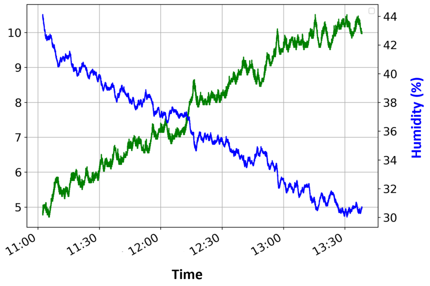


Figure 8: Temperature and humidity sensor data distribution

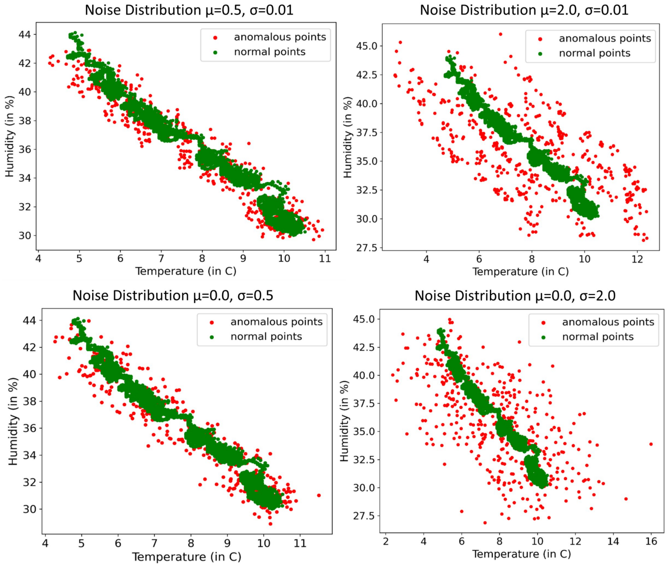


Figure 9: Gaussian noise-contaminated anomalies on sensor data

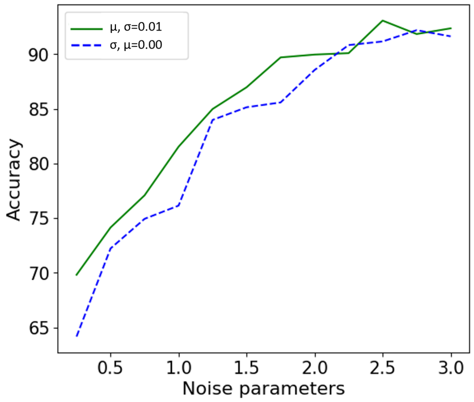
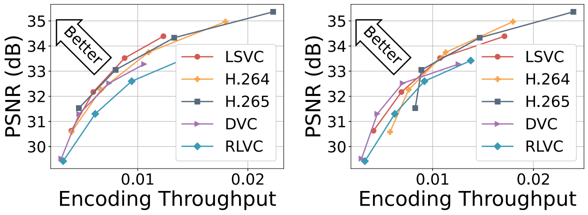


Figure 10: Accuracy of Isolation Forest Algorithm for varying noise parameters (μ and σ)

compressive offloading has the asymmetric encoder-decoder design which enables the client device (a low-end IoT device) to run a very light-weight encoder, the speed of the encoder from compressive offloading outperforms JPEG on the IoT device (a Raspberry Pi 4). Since the edge device is equipped with a GPU, it can still run the relatively heavy-weight deep neural network decoder very fast, which outperforms the speed of the JPEG decoder. Compressive offloading also achieves a higher compression ratio than JPEG. The end-to-end latency of the compressive offloading is only 42.46% of the latency of JPEG.

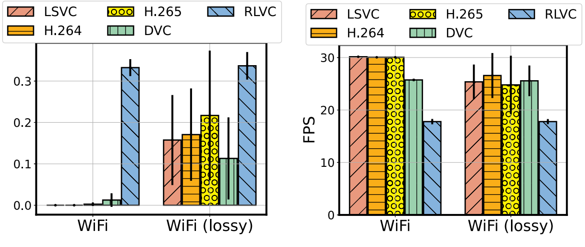
Video Compression. We compare our video compression approach, LSVC, to four baselines categorized as traditional video codecs (H.264 [28] and H.265 [24]) and learned video codecs (DVC [15] and RLVC [30]). For H.264 and H.265, they adopt the "LDP very fast" mode and the quality parameter Q=15,19,23,27 as configured in DVC [15].Figure 11(a) shows the live coding efficiency of our approach and baselines. Our approach performs competitively to H.264, H.265, and DVC when the encoding throughput is low (less than 0.01 bit/s) and significantly outperforms others when the encoding throughput is higher. Figure 11(b) (over "WiFi (Lossy)") compares the live coding efficiency of our approach and baselines. The live coding efficiency of our approach is slightly outperformed by DVC when the encoding throughput is low. As the encoding throughput increases, the live coding efficiency of our approach improves and is comparable to H.264 and H.265. Figure 12 presents the live streaming metrics, the rebuffering rate, the frame rate, and the start-up latency of our approach compared to baselines. In Figure 12(a) over "WiFi", our approach achieves a rebuffering rate of roughly 0 like H.264 and H.265. The rebuffering rate of RLVC, being over 0.3, is much higher than other baselines. In Figure 12(b) over "WiFi", our approach, H.264 and H.265 can all achieve a frame rate of 30 fps in live streaming. The frame rate of DVC is roughly 25 fps, and that of RLVC is no higher than 20 fps. Figure 12(a) over "WiFi (lossy)" shows the rebuffering rate of various approaches. The rebuffering rates of LSVC, H.264 are H.265 significantly increased by 0.1-0.2 while the rebuffering rate of RLVC is slightly affected. Figure 12(b) over WiFi (lossy) shows that the frame rates of LSVC, H.264, and H.265 are roughly reduced by 5 fps while the frame rates of DVC and RLVC are almost not affected.



(a) WiFi

(b) WiFi(Lossy)

Figure 11: The live coding efficiency of LSVC and baselines under two network conditions



(a) Rebuffering Rate

(b) Frame Rate

Figure 12: Live streaming metrics of LSVC and baselines tested on the UVG dataset. The rebuffering rate of LSVC is close to 0 like H.264 and H.265. The frame rate of LSVC reaches 30 fps like H.264 and H.265

Table 1: Comparison of compressive offloading and JPEG

	Compressive Offloading	**JPEG**
Encoding Time Cost	42.78 ms (6.79 ms on edge server)	48.88 ms (5.64 ms on edge server)
Decoding Time Cost	2.20 ms	6.84 ms
Network Transmission Latency	71.88 ms	219.53 ms
Compressed Image Size	184 KB (2.18%)	562 KB (6.65%)
End-to-end Latency	116.86 ms	275.25 ms

5 CONCLUSION

Designing a scalable and resilient system for distributed IoT applications is challenging due to a plethora of choices. Through DARTS, we present an architecture composed of different services for application deployment across infrastructure (portability), data transfer across applications, and AI-driven system capabilities. For each of these services, we present our adopted choices along with detailed experiments to highlight their pros and cons. For AI-driven services, we presented the benefits of integrating video compression, computation offloading, and data integrity in DARTS. Through DARTS, we demonstrate a wide-scale distributed infrastructure that maximizes the interoperability and flexibility required by heterogeneous IoT real-time applications.

ACKNOWLEDGMENTS

Research reported in this paper was sponsored in part by the DEVCOM Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196 (ARL IoBT CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S.Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. We thank Robert Brice, Erick Reynoso, Mark Spychala, Gordon MacDonald, Pieter Haines, Jeff Swanson, Ed Creegan, and Sean D'Arcy for their help during the course of the DARTS design.

REFERENCES

[1] A. Al-Ali, I. A. Zualkernan, M. Rashid, R. Gupta, and M. Alikarar. A smart home energy management system using iot and big data analytics approach. *IEEE Transactions on Consumer Electronics*, 2017.

[2] A. S. Al Rawahi, K. Lee, J. Robinson, and A. Lotfi. Enabling exclusive shared access to cloud of things resources. TOPIC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[3] J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier. An energy efficient iot data compression approach for edge machine learning. *Future Generation Computer Systems*, 2019.

[4] A. Banks and R. Gupta. Mqtt version 3.1.1. oasis standard. 2014.

[5] R. Buyya and S. N. Srirama. *Internet of Things (IoT) and New Computing Paradigms*. 2019.

[6] J. A. C. Cabré, D. Precup, and R. Sanz. Horizontal and vertical self-adaptive cloud controller with reward optimization for resource allocation. In *2017 International Conference on Cloud and Autonomic Computing (ICCAC)*, 2017.

[7] M. A. A. da Cruz, J. J. P. C. Rodrigues, P. Lorenz, V. V. Korotaev, and V. H. C. de Albuquerque. In.iot—a new middleware for internet of things. *IEEE Internet of Things Journal*, 2021.

[8] Grafana Labs. Grafana - the open platform for analytics and monitoring. 2022.

[9] R. Gupta, K. Nahrstedt, N. Suri, and J. Smith. Svad: End-to-end sensory data analysis for iobt-driven platforms. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, 2021.

[10] J. Koh, S. Sandha, B. Balaji, D. Crawl, I. Altintas, R. Gupta, and M. Srivastava. Data hub architecture for smart cities. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery.

[11] J. Kreps. Kafka : a distributed messaging system for log processing. 2011.

[12] R. Kridalukmana, A. F. Rochim, and F. Ramezani. Iot microservice architecture for iotaas device users. In *2021 International Conference on High Performance Big Data and Intelligent Systems (HPBD IS)*, 2021.

[13] S. Li, J. Chen, H. Yu, Y. Zhang, D. Raychaudhuri, R. Ravindran, H. Gao, L. Dong, G. Wang, and H. Liu. Mf-iot: A mobilityfirst-based internet of things architecture with global reach-ability and communication diversity. In *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2016.

[14] S. Liu, S. Yao, J. Li, D. Liu, T. Wang, H. Shao, and T. Abdelzaher. Giobalfusion: A global attentional deep learning framework for multisensor information fusion. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2020.

[15] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, and Z. Gao. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[16] T. Lu, W. Xia, X. Zou, and Q. Xia. Adaptively compressing IoT data on the resource-constrained edge. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, June 2020.

[17] H. Ma, B. Ghojogh, M. N. Samad, D. Zheng, and M. Crowley. Isolation mondrian forest for batch and online anomaly detection. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020.

[18] A. Mercat, M. Viitanen, and J. Vanne. Uvg dataset: 50/120fps 4k sequences for video codec analysis and development. In *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020.

[19] M. Nasar and M. A. Kausar. Suitability of influxdb database for iot applications. *International Journal of Innovative Technology and Exploring Engineering*, 2019.

[20] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 2017.

[21] S. S. Sandha, J. Noor, F. M. Anwar, and M. Srivastava. Time awareness in deep learning-based multimodal fusion across smartphone platforms. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2020.

[22] C. Sarkar, A. U. Nambi S. N., R. V. Prasad, A. Rahim, R. Neisse, and G. Baldini. Diat: A scalable distributed architecture for iot. *IEEE Internet of Things Journal*, 2015.

[23] S. Sotiriadis, N. Bessis, C. Amza, and R. Buyya. Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling. *IEEE Transactions on Services Computing*, 2019.

[24] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12), 2012.

[25] A. Taivalsaari and T. Mikkonen. A taxonomy of iot client architectures. *IEEE Software*, 2018.

[26] P. Tsiachri Renta, S. Sotiriadis, and E. G. Petrakis. Healthcare sensor data management on the cloud. In *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, ARMS-CC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[27] S. Wang, Y. Hou, F. Gao, and S. Ma. A novel clock synchronization architecture for iot access system. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, 2016.

[28] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7), 2003.

[29] J. Wu, Q. Liang, and E. Bertino. Improving scalability of software cloud for composite web services. In *2009 IEEE International Conference on Cloud Computing*, 2009.

[30] R. Yang, F. Mentzer, L. Van Gool, and R. Timofte. Learning for video compression with recurrent auto-encoder and recurrent probability model. *IEEE Journal of Selected Topics in Signal Processing*, 2020.

[31] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020.