Unlocking Efficiency: Understanding End-to-End Performance in Distributed Analytics Pipelines

Abel Souza, Nathan Ng, Tarek Abdelzaher, Don Towsley, Prashant Shenoy

***Abstract*—Internet of Things (IoT) devices have proliferated across a wide range of smart environments that generate vast amounts of data, necessitating the emergence of distributed edge-cloud infrastructures. However, these IoT-edge-cloud infrastructures encounter several challenges in providing efficient and effective services to users, such as real-time service delivery, robustness and resilience, and efficient deployment. Moreover, recent advancements in machine learning, particularly within the realm of deep learning, have ushered in a new era for IoT applications. These applications increasingly lean on data-intensive models to perform a multitude of functions, including classification, detection, data analytics, and decision-making. A key aspect is that many of these tasks exhibit sensitivity to latency and hinge on the deployment of models at the network's edge and on how to efficiently handle data-intensive workloads, especially when network conditions are constrained. In response to these challenges, we present an analytical discussion that delves into the intricacies of distributed IoT pipelines and workloads deployed across both edge and cloud computing environments.**

I. Introduction

The advent of machine learning models, particularly deep neural networks (DNNs), has yielded substantial enhancements in the performance of IoT applications, spanning a wide array of tasks. Within these applications, sensors increasingly rely on machine learning models to carry out functions such as classification, detection, and decision-making. However, despite notable advancements in model architecture and design, the substantial processing power and energy requirements of these models often exceed the capabilities of individual sensors or sensor nodes.

To overcome this limitation, sensors frequently resort to data offloading, transferring their data to nearby edge computing sites that offer the requisite computational capacity [1]. While edge computing initially emerged as a solution primarily geared toward latency-sensitive applications, it has since evolved into a crucial IoT processing component, especially in scenarios where privacy concern is paramount, and leveraging public cloud infrastructure is not a viable option due to not only performance constraints, but also regulatory. In this context, distributed inference pipelines are used to compute complex computations that combine data from various sources and locations to help with decision-making queries, with complex interrelated factors. The decision-making process is usually defined as a multi-stage endeavor that unfolds across different tiers within the sensor-to-decision pipeline. For the sake of enabling real-time decision-making, it becomes imperative to minimize the worst-case response time, often referred to as tail latency, throughout this IoT pipeline. Numerous optimization methods have emerged, focusing on individual components of the application, all with the overarching goal of curtailing tail latency. However, it is worth noting that the intricate nature of distributed request processing introduces dynamic variabilities such as request reordering and potential performance bottlenecks at various components, which can, in turn, escalate the end-to-end tail latency.

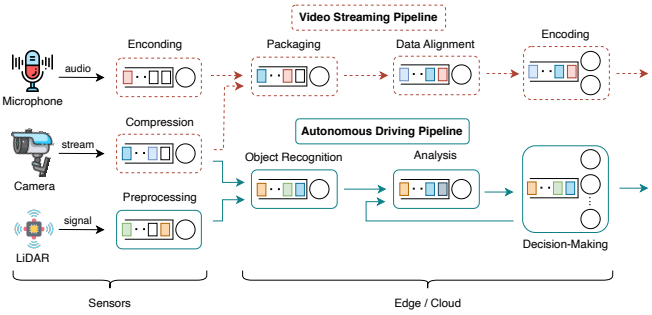

Fig. 1: *Common Distributed IoT-edge-cloud Pipelines.*

Figure 1 illustrates a standard IoT-edge-cloud architecture in which sensors establish connections with edge sites and transfer their data to either a singular machine learning model or a sequential pipeline of such models. These pipelines consist of multiple stages that can span various infrastructural layers and sites, which is particularly noteworthy in the context of IoT platforms. For instance, a video streaming pipeline entails four stages of processing for requests, whereas an autonomous driving pipeline might involve even more stages. The strategic placement of these models on edge sites is carried out with a keen awareness of resource constraints, all while striving to achieve the dual goals of minimal latency and meeting the stipulated service level objectives (SLO). This complexity is amplified by the dynamic nature of the resources available to these applications and the stringent security measures in place. Each layer, or tier, within these pipelines presents distinct constraints, encompassing considerations of performance (including latency and throughput), network bandwidth, energy efficiency (especially concerning edge devices), and the precision of predictions during inference processes.

Furthermore, a control plane plays a pivotal role in orchestrating resource allocation, determining how many resources are allocated, when, and where they are allocated. Managing these intricacies becomes especially challenging when striving for dependable performance. Requests traversing these pipelines traverse multiple queues, each representing a different stage in the process. Take, for instance, the scenario of image and LiDAR sensing, where requests progress through stages like detection and pre-processing, generating

subsequent requests such as object detection and identification, forming a cascading chain of processes. In addition to the multiplicity of queues, other variables further complicate performance management. Workload variability, along with the array of security measures in place, all contribute to the intricate tapestry of challenges in optimizing end-to-end performance within IoT distributed inferencing architectures. Finally, many of these tasks are highly sensitive to latency, relying heavily on the deployment of models at the network's edge and on the efficient management of data-intensive workloads, especially in situations where network conditions are suboptimal.

In light of these challenges, in this paper, we present an analytical examination that explores the complexities inherent in distributed IoT pipelines and workloads, encompassing deployments across both edge and cloud computing environments. Moreover, this paper presents the following contributions:

- We introduce a comprehensive set of end-to-end performance requirements for a distributed IoT-edge-cloud infrastructure, specifically designed for executing distributed pipeline applications.
- We conduct an examination of distributed pipelines under various deployment settings, shedding light on their respective advantages and disadvantages as observed in real-world deployments.

II. Background

This section presents background on IoT architectures and the distributed computations involved.

A. IoT-Edge-Cloud

An IoT-Edge-Cloud architecture encompasses different network-connected sensors embedded in various devices, such as autonomous vehicles, weather monitoring systems, smart appliances, wearables, and surveillance equipment [2], [3], [4], [5]. Because of the diversity of devices and their various sensing capabilities, applications operating over these infrastructures pose several challenges. Given the decentralized nature of IoT networks, key requirements for heterogeneous applications in dynamic environments include high scalability, real-time processing with guaranteed response times, reliability, and uninterrupted service delivery with zero downtime [6], [7]. Achieving these goals can be made possible through the implementation of a decentralized sense-compute-action infrastructure, ensuring seamless end-to-end services, ubiquitous network connectivity between subsystems, and highly available services through the use of replication and cloud services.

B. Distributed Workloads

Most distributed applications have transitioned away from monolithic architectures, where all functionalities were encompassed within a single process. Today, developers gravitate toward a microservices approach, fragmenting the application into discrete services, each dedicated to a specific function and capable of independent scaling [8]. For instance, a distributed inference application might comprise a sensor that accepts samples for recognition and an edge server for inference comparisons [9]. Consequently, requests typically traverse a network of these microservices to complete execution.

For distributed workloads, response time ranks among the main metrics applications aim to minimize. It is not solely about reducing average latency but also mitigating tail latency, exemplified by metrics like the 99th percentile latency, as this is the main metric companies often establish Service Level Agreements with their customers [10]. This dynamic fuels research endeavors dedicated to the exploration of distributed applications and the formulation of systems aimed at response time optimization. For instance, there exists a body of work dedicated to request scheduling, orchestrating the execution of requests to reduce latencies [11], [12]. Complementing this effort, much research engages in mathematical modeling to gain a comprehensive understanding of these intricate systems [13], [14], [15].

C. Edge Processing

Edge processing is a key component of IoT architectures because it offers access to computing and storage resources right at the network's proximity where applications are deployed, allowing the provision of low-latency services to users. Initially, the focus of edge computing was on streaming services such as Content Delivery Networks (CDNs), aiming to offer readily accessible data with minimal latency while conserving network bandwidth [16]. In recent times, however, the scope of edge computing has widened significantly because of the prevalence of IoT sensors. It now encompasses computing services tailored to support resource-constrained IoT sensors. Moreover, edge computing plays an indispensable role in deploying sensors in remote, hard-to-reach locations and serves as the go-to solution for privacy-sensitive services where public cloud deployments are unfeasible.

D. Performance Adaptation

Performance adaptation is a commonly utilized strategy when contending for resources in challenging scenarios such as in the edge. For instance, video-streaming applications often employ performance adaptation techniques to enhance user experience under poor network conditions [17]. In such situations, these applications may choose to reduce video quality or employ alternative frame encoding methods compared to the standard or baseline settings, ensuring a smoother and more seamless streaming experience for users. This adaptive approach also comes into play when dealing with dynamic workloads, such as unexpected spikes in demand. Moreover, performance adaptation proves to be a valuable tool in achieving alternative performance settings, enabling a system to progress even in the face of high turnaround times. Typically, this property comes at the cost of a reduced quality of service. An example of this is in network configurations, where a primary high-speed route is complemented by a slower backup route that becomes active if the primary route experiences performance issues. Finally, performance adaptation proves

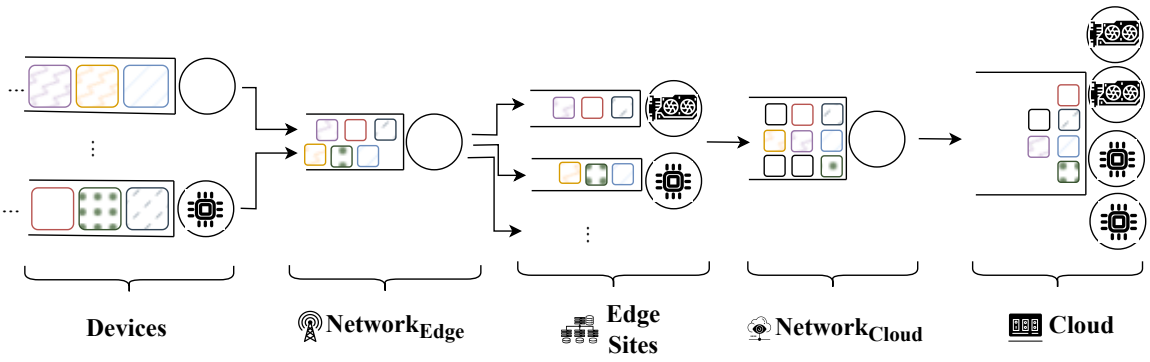

Fig. 2: *Distributed Analytics Pipelines as Tandem Queues: The overall system compromises several tiers and components, each including different sets of resource capacity, workload rate, and dynamic availability.*

to be an important factor in enhancing service reliability and maintaining system availability even with partial failures.

III. IoT-Edge-Cloud: Deployments

In this section, we present an IoT-edge-cloud system that will be used to study the performance tradeoffs of distributed pipelines. We begin with an overview of the system requirements and subsequently describe the key components.

A. Deployments

In Distributed IoT deployments, multi-staged pipelines are executed where different components and versions of an application operate across distinct tiers. In this context, "applications" refer to the business logic that includes the entirety of the distributed end-to-end pipeline, and comprises different layers. For simplification, Figure 1 distills it into three primary layers: Sensors – i.e., the devices where the samples are created –; the Edge – where certain application components may also reside –; and the cloud – typically home to the more resource-intensive, complex application components.

As the processing moves farther from the sensors, various constraints come into play, including inference latency, bandwidth limitations, accuracy, energy constraint, and throughput. Particularly during the inference phase, requests in a Distributed Machine Learning system may traverse multiple queues, each representing a distinct stage in the distributed inference process. This dynamic is akin to the flow of data sensing and pre-processing, which subsequently leads to subsequent types of requests, such as object detection, followed by identification, and so forth. In addition to the intricacy of multiple queues, other factors come into play, affecting system performance. These include workload variability, characterized by fluctuations in demand, and the presence of specific security measures and features that are inherently enabled by default.

B. Splitting Request Processing

A primary research focus for reducing latency in request processing within a distributed IoT framework involves optimizing processing distribution between sensors and edge servers. Given that sensors typically have lower processing power than edge servers, the most intuitive approach is to route input directly to the edge for processing. Nevertheless, due to constrained bandwidth between sensors and the edge, the network overhead incurred in transmitting the input may offset the processing efficiency gains at the edge, potentially resulting in decreased performance. Various research work has proposed a layer-wise DNN partitioning approach to distribute the processing between sensors and edges [18], [19]. By processing a subset of layers and transmitting the intermediate output to the edge for further processing, it significantly reduces network costs while preserving the same accuracy level. Moreover, current research endeavors to further decrease latencies by strategically balancing accuracy trade-offs. Specifically, techniques including intermediate data quantization, model compression, and vertical layer splitting, have been proposed to further minimize the network cost and the end-to-end latency by trading off less than 1% of overall accuracy [20], [21], [22], [23].

C. Distributed Applications

Modern distributed applications no longer use monolithic architectures that rely on scaling by replicating the entire application across multiple servers. Given their widespread geographic distribution, developers now opt for a software architecture that divides application functionality into distinct services, such as caching and database services. For instance, Distributed Inference and Machine Learning workloads can be conceptualized as pipelines, each comprising multiple stages that can be separated into different components and deployed across diverse infrastructural layers. For example, training processes can take place in the cloud, where ample resources are available, while inference tasks (i.e., predictions) can be executed at the edge due to stringent low-latency requirements. This architectural paradigm becomes even more significant in the context of distributed IoT platforms, characterized by the dynamic allocation of resources to applications. In response to these evolving challenges, we introduce an architecture designed to enhance and facilitate testing and experimentation within a distributed IoT network testbed.

IV. Modeling Distributed Pipelines

Effective resource allocation plays a pivotal role in ensuring optimal application performance. However, once this allocation is established, the strategies governing resource distribution and request scheduling across the system become equally crucial in maximizing performance. A complex distributed IoT scenario encompasses various components, including stream processing, seismic sensing, LiDAR fusion, analytics pipelines, video analytics, etc., all of which require careful attention to potential bottlenecks to meet the specified requirements (see Figure 1). Special consideration is needed for resource allocation in data processing-intensive tasks, ensuring optimal resource matching, especially in resource-constrained environments like the edge. As such, meeting the requirements of a distributed pipeline involves several critical aspects. A model should necessarily answer the following key questions:

- Firstly, addressing the end-to-end *throughput* requirements of distributed pipelines is key as simultaneous requests are submitted, necessitating sufficient processing capacity to prevent system overload and degradation.
- Secondly, careful management of the end-to-end *latency* requirements should be in place.
- Finally, an essential challenge is understanding *resource allocation* within specific constraints, determining where and how many resources to allocate to sustain throughput while keeping latency under control.

A. Distributed Pipelines

Consider Figure 2, which illustrates several distributed pipelines comprising multiple components. As a request progresses through such a distributed pipeline, it traverses a sequence of k components, each responsible for processing a specific type of request. As illustrated in Figure 1, these components could handle various tasks such as audio packet encoding or LiDAR sampling preprocessing. What is crucial to note is that although each component individually performs some partial processing before forwarding the request to the next component in the sequence, the successful completion of the entire execution sequence is what accomplishes the overall pipeline operation. It is worth mentioning that different requests may experience varying processing times at each component and can even follow different processing paths based on the objective of the pipeline operation, often determined by a decision-making entity (Figure 1).

B. Queueing Theory

Generally, a distributed application can be treated as a tandem $G/G/c$ network. Here, the first G means a request arrival process with a general distribution, like the exponential distribution. The second G represents the service time distribution, and c stands for the number of resources available in a server. Assuming perfect load balancing within single servers, we can model the overall performance of an end-to-end pipeline as a $G/G/1$ queue and represent the c server resources as if they were a single unit. If the server becomes overloaded, requests start queuing up, resulting in waiting times. The waiting time W_{n+1} for a new request $n+1$ can then be expressed as:

$$W_{n+1} = max(0, W_n + s_n - a_n), \tag{1}$$

where W_n and s_n denote the waiting time and service time for the previous request n, respectively, and a_n represents the inter-arrival time between requests n and $n+1$.

The system utilization ρ_k for server k indicates the fraction of time the server is occupied and can be defined as

$$\rho_k = \frac{\lambda_k}{\mu_k}, \tag{2}$$

where λ_k and μ_k represent the request arrival rate and service rate at server k, respectively.

A tandem queue network is considered stable when ρ_k is less than 1 across all tiers. If ρ_k exceeds 1, it indicates insufficient server capacity, whether due to poor resource allocation or spikes in workload rate. Either way, it results in excessively long response times — a condition known as saturation that should be avoided in distributed systems.

C. Implications

The model described above serves as a valuable tool for comprehending the functioning of distributed systems on a large scale. It is applicable to any system ranging from a single application to a complex IoT-edge-cloud architecture supporting various distributed pipelines. The main challenge lies in minimizing the average and tail performance – measured as response time and overall throughput – during peak periods to minimize wastage of resource capacity [24], [25]. This is because enforcing system-wide policies in distributed systems is particularly challenging due to some types of complexity. The first type is horizontal, where applications can have hundreds of components [26]. A user request may trigger numerous additional requests, which are served independently by a local, single component. The second type of complexity is vertical, where each component dispatches additional requests to a thread pool, and each thread is considered as an independent entity by the scheduler. Finally, given these complexities, few solutions consider the distributed system holistically, with much research using system-level optimizations to aggressively improve the tail performance.

V. Evaluation

In this section, we showcase the untapped potential for optimizing request execution within distributed IoT pipelines.

A. Experimental Setup

Our analysis comprises a simple application with a 2-stage pipeline. We use an open-loop workload generator that produces requests with inter-arrival times following the Poisson distribution. Each request's target service time at each stage is distributed exponentially. The generator forwards the request to the first application stage through a client TCP connection. On the server side, upon receiving a new request, the server

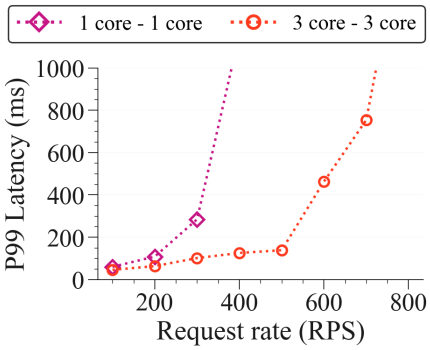

Fig. 3: *Tail Performance under different resource allocation schemes. A naive approach can optimize system performance but may result in resource over-allocation.*

creates a thread to execute an idle loop, emulating the service time predefined in the request. After completing the idle loop, the request is forwarded to the next tier, continuing until it reaches the last tier. Finally, the request is sent back along the original path to the client, where we measure the performance metrics.

B. Optimizing Latency and Resource Allocation

Figure 3 highlights the necessity of a model-driven resource allocation algorithm for minimizing latency while avoiding resource wastage. In this analysis, we vary the number of cores at each pipeline stage and measure the end-to-end tail latency. When each component operates with only one core, as illustrated by the purple line, we observe a significant spike in P99 latency at a request rate of 300 rps, which surpasses one second at 350 rps, indicating system saturation. To enhance performance, a naive approach is to increase the resources of all components, for instance, to 3 cores per component, as illustrated by the orange line. While this effectively reduces latencies, it may lead to over-allocation of resources since real-world applications often exhibit varying workload rates with varying service time distributions across components, resulting in a complex – possibly dynamic – resource allocation needs. In such cases, a model-driven approach becomes invaluable, enabling us to pinpoint performance bottlenecks within the pipeline and adjust resource allocation accordingly, such that we can navigate between the design space among the two lines in Figure 3. For instance, we can identify and allocate more resources to components experiencing higher queueing delays while maintaining the current resource allocation for other unsaturated components so long as Equation 2 does not exceed 1.

Key Insight: *Resource flexibility allow us to meet application requirements, including mean and tail latency constraints for requests, all while minimizing resource usage.*

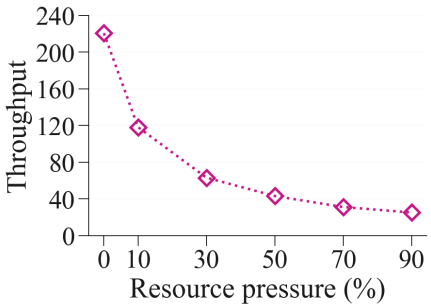

Fig. 4: *Infrastructural Performance under resource pressure: As the external resource pressures in the pipeline components increase, resource contention reduces application throughput.*

C. Impact of Resource Pressure

We then assess how resource pressure impacts application throughput in typical IoT settings. Both devices and edges in these scenarios are resource-constrained, making their performance susceptible to external resource pressure factors such as kernel power management threads and garbage collection. These non-deterministic factors disrupt the normal execution of requests, consequently reducing the overall system throughput. We illustrate these effects in Figure 4, where we emulate resource pressure levels while keeping the request rate constant. We observe that the higher the resource pressure, the lower the system throughput. This underscores the necessity for load-balancing and routing algorithms to determine the optimal component in the pipeline (e.g., which edge server) for routing the next request to avoid and mitigate the negative effects of resource pressure. A potential solution entails monitoring the load and detecting queueing delay spikes at each component. This allows us to identify and analyze the root causes pertaining to why certain components experience resource pressure. Subsequently, we can dynamically adjust the resource allocation, and alternatively use re-routing techniques to divert traffic away from the congested component until it returns to normal operating conditions.

Key Insight: *Effective management of distributed pipelines requires seamless integration with the underlying resource management layers. This integration is essential for mitigating the performance impacts caused by external workloads that exert resource pressure on the infrastructure.*

D. Challenges of Proximity-Based Routing

To minimize the network delay between components in the pipeline, the most intuitive approach is to route the request to the closest edge server. However, existing literature suggests that this strategy may not necessarily yield improved performance due to workload imbalances and the resource constraints typical of edge computing [27]. This phenomenon is called *edge-inversion*. We illustrate a similar observation in Figure 5 within a pipeline. After processing at the first component, while forwarding the request to

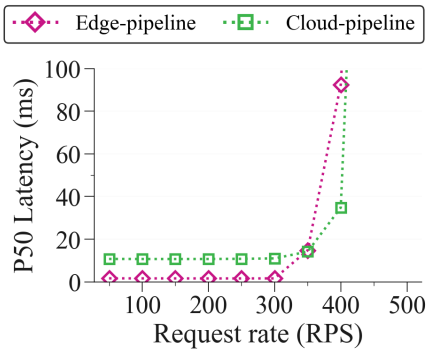

Fig. 5: *The impact of edge inversion: Despite the closer proximity of edge servers compared to the cloud server, high system utilization can result in queueing delays that offset the network cost savings.*

the nearest component (i.e., the edge) reduces network costs, this advantage is overshadowed by queueing delays at the edge during periods of high utilization (e.g., at 400 rps).

Key Insight: *Distributed IoT resource managers can improve the system and application performance by utilizing model-driven techniques that determine the optimal resource allocation and placement while minimizing latency.*

VI. Challenges and Future Work

Designing a scalable and robust system for distributed IoT applications poses significant challenges, primarily because of the multitude of available options and decisions to be made.

A. Challenges

First, determining the optimal placement of pipeline stages across a distributed infrastructure, all while preserving the quality of service is a complex challenge. This is exacerbated by the continually shifting dynamics of the infrastructure, the diversity and heterogeneity of the nodes involved, and the scalability limitations of the models in use. Furthermore, the trade-offs between accuracy, latency, and sensing rate can vary significantly depending on the specific scenario and the use case of the application.

As mentioned, the dynamics within the infrastructure's resource availability need to be comprehended holistically. This involves not only understanding the workload and optimizing its resource allocation, with an emphasis on highlighting specific distributed pipeline requirements, but also doing these online and at runtime, which can be extremely challenging assuming that even any two users operating similar pipelines may – from the point of view of the infrastructure – need different and contradicting resource requirements. Additionally, the security aspect cannot be overlooked. Attackers often induce resource pressure through hidden background tasks that sniff and query the system for benign tasks, which directly impacts the performance of the pipeline. Assuming that all other factors remain balanced, including workload rate and optimal resource allocation, the identification of such attacks can be accomplished by monitoring pipeline tail performance.

B. Future Work

Efficiently modeling these intricate end-to-end pipelines necessitates various approaches, including the application of machine learning models and queueing theory. However, it is important to note that while machine learning models are highly effective in optimizing mean latency, their performance tends to be less robust when it comes to addressing tail latency issues. Some pipelined operations are very sensitive to performance degradation. In particular, tail latency demands a focused approach, as any degradation in this metric can have adverse consequences for various reasons.

Queueing Theory emerges as an intuitive and extensively researched framework for understanding these aspects, particularly within the context of multi-staged workloads in IoT. As a modeling framework, queueing models offer valuable insights by providing stochastic performance boundaries concerning response times, encompassing metrics such as averages, variances, distributions, and tail behavior. Furthermore, it empowers us to estimate worst-case performance scenarios, subsequently enabling the development of policies for request prioritization and processing. Additionally, it aids both users and developers in making informed decisions across diverse scenarios. As such, integrating such models into resource managers would potentially improve the overall system performance and robustness. While queueing models offer an appealing and fundamental means of defining distributed systems that can be derived from various profilable metrics, the majority of solutions are presented as closed-form expressions. Consequently, it is crucial to exercise caution when proposing adaptive solutions, as resource management systems should be flexible enough in order to handle *unknown-unknowns* [28] scenarios that may not be adequately captured by these models [29].

References

[1] R. Gupta, B. Chen, S. Liu, T. Wang, S. S. Sandha, A. Souza, K. Nahrstedt, T. Abdelzaher, M. Srivastava, P. Shenoy, *et al.*, "Darts: Distributed iot 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*, pp. 15–23, 2022.

[2] V. Va, T. Shimizu, G. Bansal, and R. W. Heath Jr. 2016.

[3] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, and N. D. Lane, "Embench: Quantifying performance variations of deep neural networks across modern commodity devices," in *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, EMDL '19, (New York, NY, USA), p. 1–6, Association for Computing Machinery, 2019.

[4] S. Wang, A. Pathania, and T. Mitra, "Neural network inference on mobile socs," *IEEE Design & Test*, vol. 37, no. 5, pp. 50–57, 2020.

[5] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool, "Ai benchmark: All about deep learning on smartphones in 2019," 10 2019.

[6] J. Dean and L. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, pp. 74–80, 02 2013.

[7] M. Mayer, "What Google Knows," *Proceedings of the Third Annual Web*, vol. 2, 2006.

[8] Z. Zhang, M. K. Ramanathan, P. Raj, A. Parwal, T. Sherwood, and M. Chabbi, "CRISP: Critical path analysis of Large-Scale microservice architectures," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, (Carlsbad, CA), pp. 655–672, USENIX Association, July 2022.

[9] S. Yao, Y. Zhao, A. Zhang, S. Hu, H. Shao, C. Zhang, L. Su, and T. Abdelzaher, "Deep learning for the internet of things," *Computer*, vol. 51, no. 5, pp. 32–41, 2018.

[10] M. Belshe, "More Bandwidth Doesn't Matter (Much)," 2010. http://bit.ly/2TxHv84.

[11] K. Kaffes, T. Chong, J. T. Humphries, A. Belay, D. Mazieres, and C. Kozyrakis, "Shinjuku: Preemptive Scheduling for μsecond-scale Tail Latency," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pp. 345–360, 2019.

[12] A. Ousterhout, J. Fried, J. Behrens, A. Belay, and H. Balakrishnan, "Shenango: Achieving High CPU Efficiency for Latency-Sensitive Datacenter Workloads," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pp. 361–378, 2019.

[13] O. J. Boxma and M. Vlasiou, "On Queues with Service and Interarrival Times Depending on Waiting Times," *Queueing Systems*, vol. 56, no. 3-4, pp. 121–132, 2007.

[14] M. Berg and M. Posner, "On the Regulation of Queues," *Operations research letters*, vol. 4, no. 5, pp. 221–224, 1986.

[15] D. Towsley and F. Baccelli, "Comparisons of Service Disciplines in a Tandem Queueing Network with Real Time Constraints," *Operations Research Letters*, vol. 10, no. 1, pp. 49–55, 1991.

[16] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, p. 2–19, aug 2010.

[17] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[18] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Archit. News*, vol. 45, p. 615–629, apr 2017.

[19] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, p. 1423–1431, IEEE Press, 2019.

[20] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: Synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[21] S. Yang, Z. Zhang, C. Zhao, X. Song, S. Guo, and H. Li, "Cnnpc: End-edge-cloud collaborative cnn inference with joint model partition and compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, p. 4039–4056, dec 2022.

[22] S. Tuli, G. Casale, and N. R. Jennings, "Splitplace: Ai augmented splitting and placement of large-scale neural networks in mobile edge environments," *IEEE Transactions on Mobile Computing*, vol. 22, pp. 5539–5554, sep 2023.

[23] Y. Gao, W. Wang, D. Wang, H. Wang, and Z. Zhang, "Cloud-edge inference under communication constraints: Data quantization and early exit," in *2022 International Symposium on Wireless Communication Systems (ISWCS)*, pp. 1–6, 2022.

[24] J. Leverich and C. Kozyrakis, "Reconciling High Server Utilization and Sub-millisecond Quality-of-service," in *European Conference on Computer Systems (EuroSys)*, ACM, 2014.

[25] D. Desmeurs, C. Klein, A. V. Papadopoulos, and J. Tordsson, "Event-Driven Application Brownout: Reconciling High Utilization and Low Tail Response Times," in *Cloud and Autonomic Computing (ICCAC)*, 2015.

[26] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a Warehouse-scale Computer," *SIGARCH Comput. Archit. News*, vol. 43, June 2015.

[27] A. Ali-Eldin, B. Wang, and P. Shenoy, "The hidden cost of the edge: A performance comparison of edge and cloud latencies," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, (New York, NY, USA), Association for Computing Machinery, 2021.

[28] S. Rumsfeld and G. Myers, "Defense.gov News Transcript: DoD News Briefing (defense.gov)," 2002. http://archive.today/8k6bU.

[29] L. A. Okashah and P. M. Goldwater, "Unknown unknowns: Modeling unanticipated events," in *Proceedings of Winter Simulation Conference*, pp. 689–694, IEEE, 1994.