Principles and Practices for Application-Network Co-Design in Edge Computing

Ruozhou Yu, *Senior Member, IEEE*, Guoliang Xue, *Fellow, IEEE*

***Abstract*—Edge computing promises low-latency and high-throughput real-time processing to enable critical and life-changing future applications such as fully autonomous driving and metaverse, and is receiving wide interest from application designers, service providers and academic researchers. As for now, however, the full power of edge computing has yet been unleashed, as stakeholders face difficulty in realizing the promised services and hence investment and efforts waver. This article describes a new conceptual framework, called the *application-network co-design* approach, to align and direct efforts across application, computing and networking domains towards fully fledged edge computing. At a high level, we introduce current practices and efforts in edge computing, identify limitations of existing efforts, present the overarching goal and principles of the co-design approach, and discuss existing and future prerequisite technologies for implementing the approach. We illustrate the principles and process of the co-design approach with two compelling edge applications: autonomous driving and virtual/augmented reality for metaverse.**

***Index Terms*—Edge computing, real-time applications, computing-network convergence, application-aware networking, end-to-end performance guarantee**

I. Introduction and Background

Edge computing is a computing paradigm where computing resources, such as edge servers or micro-data centers, are deployed within edge networks such as the cellular network where end devices like mobile phones or robots are directly connected [1]. When computing is near data sources or sinks, it can significantly lower network latency and save network bandwidth for data transmission. This is in contrast to cloud computing, where computing power is remotely in the Internet core, and hence end-to-end performance is highly affected by the unstable and unpredictable Internet used to access the cloud.

The need for edge computing comes from modern applications that crucially need ultra-low latency and high throughput. Examples include autonomous driving, virtual/augmented reality (VR/AR) and smart manufacturing. Using the cloud would likely violate these applications' performance requirements, making them unusable or unsafe to use. Edge computing also improves user experience of many other applications including web services, internet-of-things (IoT), smart cities, connected healthcare, and many more.

Edge computing's goal is to support applications' stringent performance goals that cannot be satisfied by traditional cloud computing. Performance goals are commonly in terms of quality-of-service (QoS) metrics such as latency, throughput and reliability. Many edge applications specifically have *end-to-end* QoS goals. For instance, a self-driving car may require sensory processing results to be received within 100ms of data generation. This includes time spent on both computing and communications. Violating the 100ms end-to-end bound may lead to accidents or crashing due to decision-making based on stale information.

In an edge computing ecosystem, several key stakeholders exist, each with distinct goals:

Infrastructure providers: Computing and network are managed by computing and network providers respectively. They may be different parties, e.g., when a cloud provider deploys regional micro-data centers in contract with a local Internet Service Provider (ISP). In other cases, a single computing-network provider owns and manages the entire infrastructure, e.g., a cellular provider deploying mobile edge computing (MEC) in her cellular network. Their goal is to maximize revenue by providing computing/network resources that meet application needs, subject to resources owned, and deployment, maintenance and utility costs.

Application owners and users: An application owner seeks to deploy and operate her application in the edge network, in place of or complementary to the cloud. The goal is to serve users with usable and satisfactory performance. If performance degrades, the owner may lose revenue or face legal issues (such as accidents involving self-driving cars). An owner's tasks include requesting resources from infrastructure providers, configuring the application at the edge, and maintaining its operation. Their goal is to maximize or guarantee user satisfaction, subject to costs for edge deployment and operations.

Platform providers: Platform providers may arise in edge computing to orchestrate resources from multiple computing and network providers. A platform provider meets-and-matches applications' needs with providers' offerings, and earns revenue by charging service fees from one or both parties. It is not uncommon that a platform provider is also an infrastructure provider. For instance, an ISP that has contracts with multiple computing providers can take advantage of her role and act as an intermediate exchange who provisions computing services from different providers for applications. Their goal is to maximize revenue by orchestrating resources from multiple infrastructure providers.

Born as a performance-oriented paradigm, edge computing faces unique challenges to satisfy performance goals of edge applications and users. Edge computing features geo-distributed computing resources to bring computation to users, but any given location has limited resources. Thus hotspots may arise when users are crowded in certain areas. User mobility exacerbates the problem since hotspots may move over time. Furthermore, a user's perceived performance is affected by many factors. Some are controlled by providers and application owners, such as resource and application configurations. Other factors, such as wireless channel conditions, user mobility and hardware/software failures, are not controlled by any party. These make it exceptionally difficult to provide stable and guaranteed performance to edge application users. The issue is also aggravated by the diverse goals and ad hoc management of different stakeholders above. Their uncoordinated efforts could lead to ineffective performance optimization and resource wastage due to obliviousness of each other's knowledge and decision-making. The next section elaborates on this last issue from an academia perspective.

II. Current Approaches and Limitations

The community has taken diverged approaches to define how edge computing should work:

Computing perspective: One line of research spans from distributed and cloud computing. Here, edge computing is regarded as a *distributed cloud environment* (called an edge-cloud) compared to a centralized cloud. Research has focused on managing virtual machines (VMs) or containers to satisfy applications' computation needs. This aligns with the cloud paradigm where computing is the only (sensible) need to be served. Key techniques inherited from cloud computing include *auto-scaling* to tackle dynamic load, *load balancing* to reduce overheating, *failover* to handle hardware/software failures, etc.

Most efforts along this line overlook or underrate the impact of networking in edge computing. Assumptions such as access links as bottlenecks, static pair-wide bandwidth, regular network topologies or perfectly stable channels are frequently made but seldomly justified in practice. As such, existing techniques can barely provide end-to-end performance guarantees, which is similar to the cloud.

Networking perspective: Another line originates from the networking community. Here edge computing is tied to the network that it is built upon. For instance, MEC builds computing power directly within mobile networks. Existing research on MEC thus emphasizes network components and their interplay with computing, including bandwidth/radio allocation, communication scheduling, routing, mobility handling, etc. Taking communications into account enables identifying network bottlenecks and is essential for modeling end-to-end performance.

A heritage from traditional networking is the focus on *best-effort services* for aggregate traffic, and thus negligence of individual applications' or users' needs. For example, network queueing theory usually studies *average* queueing delay or *long-term* queue stability, while neglecting worst-case performance experienced by few users. Network QoS is implemented via limited priority classes (e.g., DiffServ) in coarse granularity. Moreover, application-agnostic networking lacks the ability to model end-to-end processing of user demands in distributed applications. These significantly hinder realizing end-to-end performance guarantees.

Application perspective: Finally, many efforts focused on designing applications that utilize edge computing. Content delivery, video streaming and IoT are by far the most successful,

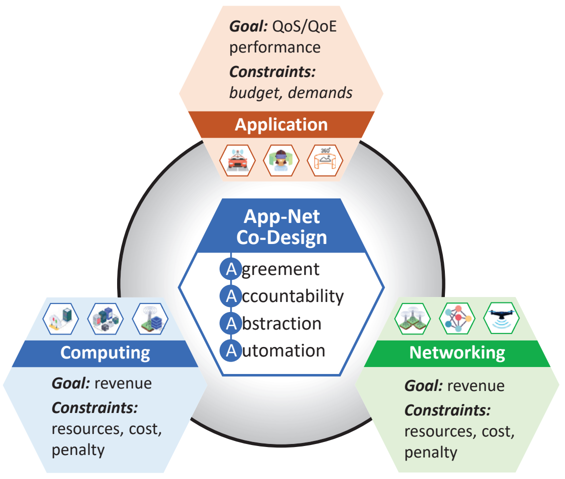

Goal: QoS/QoE performance
Constraints: budget, demands
Application
App-Net Co-Design
Agreement
Accountability
Abstraction
Automation
Computing
Goal: revenue
Constraints: resources, cost, penalty
Networking
Goal: revenue
Constraints: resources, cost, penalty

Fig. 1: Application-network co-design represents a convergence of efforts from different parties for delivering satisfactory and guaranteed edge computing performance, with four major principles: *agreement*, *accountability*, *abstraction*, and *automation*.

but many more are in progress including autonomous driving, metaverse, smart manufacturing and connected healthcare. As many edge applications are still immature, there has been limited understanding in the actual performance and cost-efficiency of these applications, and in return how applications affect design and management of the edge.

As a key limitation, current efforts mostly assume that each application exclusively occupies the entire edge, including communication channels and computing servers. This would lead to over-optimistic estimation of edge computing's costs and benefits. In some cases, this might cause unexpected failures or violation of applications' performance goals, and lead to safety issues in use cases such as autonomous driving.

From the above, the lack of coordinated knowledge and efforts across computing, networking and application domains has become the root cause of many obstacles faced by edge computing. A principled approach is in-need to remove the barrier across these domains and align their efforts.

III. The Co-Design Approach

A. Overview

Application-network co-design[1] (*a.k.a.*, the co-design approach) is a principled approach to integrate networking, computing and application domain efforts for joint edge resource allocation and performance optimization, shown in Fig. 1. In this approach, all efforts are centered around satisfying *end-to-end performance goals* of each and every accommodated edge application, subject to resource, cost, energy, overhead, topology, and other constraints posed by the application owner and edge providers. Its core lies in closely coupling applications with computing and network management through (dynamic) provisioning and service-level agreement (SLA) guarantees. The following "4A" principles guide the design and implementation of co-design solutions:

[1]As application and computing are always coupled *in practice*, we omit "computing" in naming our approach for brevity and to emphasize the convergence between application and network that differs from existing paradigms.

Agreement: Achieving end-to-end performance guarantee requires cooperation and mutual agreement among all parties. SLAs are extensively used in the Internet and the cloud to express mutually agreed performance goals, with negotiated compensation terms when requirements are violated. Traditionally, SLAs are established for per-component availability goals, e.g., "5Mbps data rate for link A for 99.9% of time". At the edge, due to stringent performance requirements, a major paradigm shift is from component-wise SLAs to **end-to-end SLAs**. For instance, an application owner may request for "100ms end-to-end latency for 99.9% of user requests with up to 500 requests-per-second". The guarantee covers the entire process from when a request leaves a user device until when computation result is received by the device, traversing territories of all parties: the edge network, computing node(s), and application software. Naturally, fulfilling the guarantee requires collaboration, coordination and accounting across all parties.

Accountability: SLAs are associated with financial interests like application owners' costs and providers' violation penalty. Hence accountability is essential. Before an SLA is negotiated, all parties need to obtain and agree on an accurate, reliable and trustworthy performance model of the target application. This ensures that all parties correctly understand: 1) if the SLA terms can be fulfilled by available resources, 2) configuration and resource allocation required to fulfill the SLA, 3) realms of responsibilities of each party for fulfillment, and 4) how to detect and attribute an SLA violation. Performance model can be obtained by profiling the application in a controlled environment, *e.g.*, a virtualized edge environment run by a provider that emulates the real environment. After an SLA is established, runtime measurement is needed to validate the SLA terms, detect violation and attribute faults. An accounting module is needed to collect, analyze and store measurement results, with a certified method for result validation documented in the SLA. Agreement and accountability (with incentives or penalty) jointly forms the basis of realizing secure and trustworthy edge environment.

Abstraction: Heterogeneity widely exists in the edge. Co-existing applications have different processing components, structures, traffic patterns, and performance goals. Furthermore, edge resources including communication channels, computing devices and virtualization technologies differ. A general abstraction is required to capture the inter-play between application structure, user demands and edge resources. Take the *realization graph* abstraction in [2] as an example. A realization graph defines essential elements for realizing an application, including: which edge nodes host each application component, how to distribute load between components, and what constitutes an end-to-end processing realization of a user request for which the end-to-end performance can be measured. Such an abstraction not only guides provisioning algorithm design, but also defines how the provisioning result is realized in the edge. The abstraction in [2] is however network-agnostic. Abstractions fully embracing the co-design approach is further needed.

Automation: The edge environment is volatile with wireless links, mobility, and demand changes. This necessitates automated responses to dynamic events, including on-demand resource provisioning and dynamic SLA establishment. Automated responses can be triggered by predicted demand changes, or explicit requests from users. For instance, an autonomous driving application can request for additional resources at intersections that may become congested; a multi-player VR game can request for a short-term SLA for each matched game initiated by its users. Machine intelligence can play key roles for tasks such as demand prediction, automated resource provisioning, and SLA negotiation. Automation is crucial for alleviating the operation and coordination overhead of our approach.

B. Key Prerequisites for the Co-Design Approach

The co-design approach requires computing services that can be dynamically and automatedly orchestrated with performance isolation, such as VMs and containers. Cloud native solutions such as Kubernetes and KubeVirt are readily available for dynamic orchestration, instance migration and auto-scaling for VMs and containers. Application profiling is a widely used technique for performance modeling and optimization [3]. These current technologies are ready to embrace the co-design approach, pending support from the other parties.

Network prerequisites: Network operations in co-design must be able to identify individual applications' traffic flows, isolate network resources (e.g., bandwidth and priority queues), and measure QoS of each traffic flow. Traditional networks are difficult to implement this approach since routers/switches have a limited number of queues, and networks are designed to be application-agnostic. Programmable and software-defined networks (SDNs) are the *de facto* approach to application-aware networking. For example, P4-supported routers/switches can be used to dynamically program the network for measurement [4], routing and traffic engineering; software-defined radio with srsRAN or alternatives can support dynamic allocation of radio resources in runtime [5]. Additionally, network measurement with ultra-fine granularity (e.g., nanosecond-level packet timestamping [6]) is seeing its way into large-scale testbeds and likely production environments soon, offering tools for accurate and intelligent network monitoring. As most of these functionalities are already implemented, additional overhead for co-design is mostly incurred in the control plane that commonly has sufficient resources in SDNs.

Application prerequisites: A *co-designed* application requires several considerations during design and implementation. First, native measurement and accounting are needed for end-to-end SLAs. This requires a built-in measurement module that can 1) track end-to-end performance of user requests, 2) obtain component-wise performance breakdown for bottleneck identification and resolution, and 3) incur minimal impact on application performance. OpenTracing and alternatives [7] represent a promising approach, enabling distributed tracing of a user request from end to end. Each application component processing a request records and reports the accurate timestamps of reception and transmission. The overhead of fine-grained measurement can be minimized by sampling only a fractions of requests for tracing, such as one in a thousand. All measurement data should be gathered at a central, trusted server, where the data is analyzed to detect any SLA violation and to make dynamic adjustments.

Second, the application should have maximal deployment flexibility and low migration overhead. A modular design with singletons or loosely coupled components allows dynamic deployment and migration in face of changing environment and dynamic load. Microservices and serverless computing are examples of suitable design paradigms, both enabling runtime adjustment of service deployment and/or inter-dependencies across services.

Third, automated management is needed for the application. This includes measurement and accounting aforementioned, but also user modeling and demand estimation, adaptive configuration such as load balancing, and interfacing with computing/networking for provisioning and negotiation. While no existing product is available to automate all these tasks, service meshes and orchestrators are good candidates upon which fully automated management modules can be implemented.

IV. Case Studies with Example Applications

Example 1: Autonomous Driving Use Case

Autonomous driving vehicle (ADV) is an important use case of edge computing. ADVs need to process sensory data and make driving decisions in real time. Vehicles further need to collaborate for safe driving. For instance, automated platooning enables a group of vehicles to drive together at the same speed; collaborative perception allows vehicles to remove blind spots via data sharing and view merging. In general, an end-to-end autonomous driving system can be decomposed into various tasks, which can be implemented across on-board units (OBUs) and the edge based on task requirements.

Consider an ADV system supporting three high-level tasks in Fig. 2(a): collaborative collision avoidance (CCA), collaborative blind spot removal (CBR), and route planning (RP). Perception, ego state management and control are basic components commonly implemented at OBUs. Components such as object tracking, localization, mapping, and motion planning can be implemented at OBUs, in the edge, or both at the same time serving different tasks. The three high-level components (CCA, CBR and RP) all rely on fusing data from multiple vehicles, and are best implemented with help of the edge.

On-board and edge components communicate via the *vehicle-to-infrastructure (V2I)* interface defined in IEEE 802.11p or 3GPP C-V2X standards [8]. Vehicles share the wireless channel via broadcast communication. Messages from different components have different sizes and rates. The high-level tasks (CCA, CBR, RP) also have different real-time requirements: CCA requires maximum end-to-end latency of 25ms, CPR requires 100ms, and RP has sub-second or higher latency bounds [9]. In a typical application-agnostic network, all messages randomly access the wireless data link, and share the same bottleneck if the channel is busy (e.g., at a crowded intersection). Computing at the edge that does not respect the limited link resource will experience poor end-to-end performance even if computation is done in real-time.

To employ the co-design approach, first a profiling step is taken jointly by the application owner and edge providers. A virtual environment is setup by providers to run offloadable components of this application. The application owner supplies sample user demands, such as via test vehicles. End-to-end tracing is enabled in profiling, and measurement data are stored and analyzed. From profiling, a performance model is derived with information on: 1) traffic volumes between on-board and edge components, 2) network latency and throughput given channel conditions and radio bandwidth, and 3) computation latency and throughput given different resource configurations. Note that profiling can be done in a privacy-preserving manner: providers need not know exact functionalities of components, but just regard components as black-box programs and observe that profiling is done in a reliable and trustworthy manner. After done, data and profiled model are stored by all parties for provisioning and auditing.

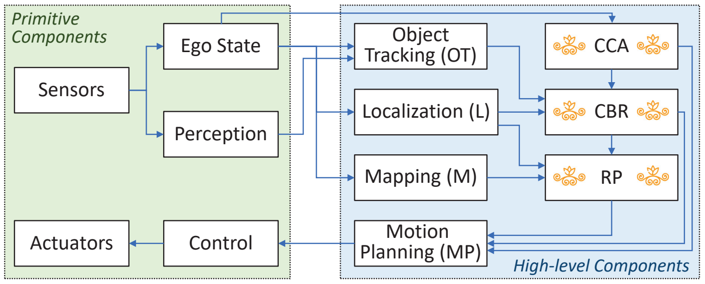
After profiling, the application owner can initiate SLA requests. A distributed optimization algorithm governs the negotiation process. In the algorithm, providers take as inputs application components and inter-dependencies and map them to computing nodes and network paths with resource allocation, generating one or multiple *realization plans*. The plans and pricing are then sent to the application owner, who inspects and picks the best plan given her end-to-end performance goals and budget. An example is shown in Fig. 2(b). The application owner can dynamically request SLAs to handle changing demands at different locations.

At runtime, distributed tracing is employed for randomly selected vehicle requests for end-to-end inspection. If SLA violation is detected, data will be sent to all parties for validation. Penalty will apply when a provider is found liable.

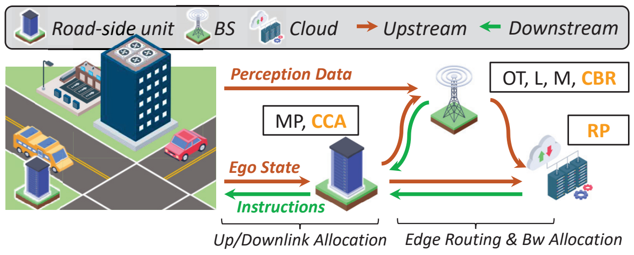
A key for the ADV use case is *risk-awareness*. Since the scenario is safety-critical, highly reliable SLA satisfaction is required, and any violation case must be handled with precaution. For instance, a vehicle should host backups for any offloaded component to tackle failures or lagging. The backups need not consume much computation power, but must be standby to tackle any emergency. Resource provisioning needs to tackle unexpected variations in communication, mobility and demand. Risk measures such as *conditional value-at-risk*, and robust approaches such as *distributionally robust optimization*, should be employed for robust provisioning.

Example 2: Multi-User VR/AR Use Case

VR/AR is another scenario where edge computing is highly desired. VR/AR applications need to process and deliver visual data to users in real-time to have acceptable user experience, and have tighter latency requirements than most other applications. An end-to-end rendering latency of over 15-30ms can already cause motion sickness to users [10]. Multi-user VR/AR, such as multi-player gaming or teleconferencing, is more chal-

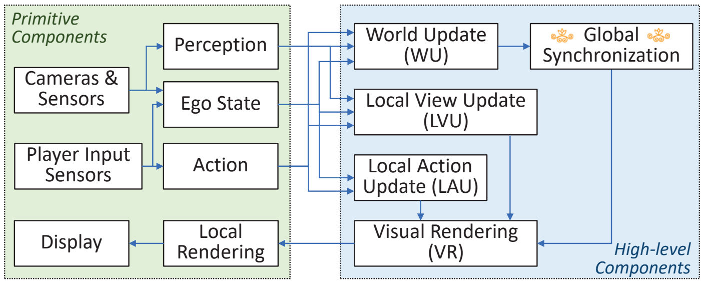


(a) An ADV system with three high-level tasks: *collaborative collision avoidance (CCA)*, *collaborative blindspot removal (CBR)*, and *route planning (RP)*.

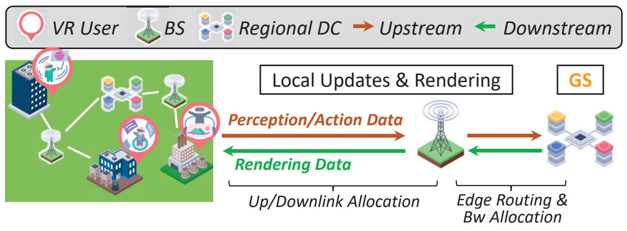


(b) An example *realization* of the ADV system with offloaded mid- and high-level tasks. In addition to task placement (computing), it also includes network routing and resource allocation.

Fig. 2: Autonomous driving vehicle (ADV) use case illustration.



(a) A multi-player VR/AR game with local and global updates.



(b) An example *realization* of the game with asynchronous local-global updates, including routing and bandwidth allocation.

Fig. 3: Virtual/augmented reality (VR/AR) use case illustration.

lenging since users are distributed and hence communications can be excessive yet tightly latency-budgeted.

Due to hardness of processing and transmitting raw video captures or game frames in real-time, many VR/AR applications employ layered processing. Specifically, user view updates are divided into different layers, processed by different components [10]. For instance, in VR gaming, one layer corresponds to game world changes such as monsters spawning or demising, which needs to be synchronized among all players but does not need ultra-low latency; another layer corresponds to a player's local changes such as facing direction change or player action, which needs immediate feedback. To improve responsiveness to action-triggered events, user behavior prediction can be employed. Processing components can be deployed across user devices, the edge and the cloud. An example is shown in Fig. 3

A co-design approach again begins with profiling with real or synthesized games or meetings, where a performance model in terms latency, throughput, resources and demands (number of users and traffic per user) is generated. Then, dynamic SLA negotiation and provisioning can take place *on-demand*. For instance, when a new matched game or meeting is initiated, a short-term SLA can be negotiated to cover exactly the period of the game or meeting. This ensures on-demand pricing and avoids resource under-utilization. Since user behaviors and traffic demands are relatively static for these applications, risk-awareness is not as crucial as in autonomous driving, except when users exhibit high mobility or dynamic wireless channels. Finally, runtime SLA validation again takes place with end-to-end tracing and measurement, and violation is punished post completion of each SLA.

V. Roads Behind and Ahead

A. Relationship to Existing Approaches

The idea behind the co-design approach is not new. It naturally inherits from concepts and paradigms such as network function virtualization (NFV), service-oriented architecture (SOA) networking, or application-aware networking (AAN) [11]. All of these seek the closer coupling between application/service, computing, and networking than traditional agnostic approaches. Nevertheless, there exists a fundamental conceptual shift from existing concepts to this new approach. In existing concepts, it is common that one domain serves as the "first-class entity" who defines the primary objective of design and optimization, and other domains are mainly in support roles. For instance, in SOA and the succeeding microservice-based architecture, the primary goal is to design software that is composed of loosely coupled or independent services, and computing and networking serve support roles whose constraints and goals are seldom considered in application design. In AAN or NFV, the primary goal is instead the optimization of network operations in terms of improved efficiency and reduced cost, while applications are regarded as given and fixed and their performance goals commonly served with "best effort".

The co-design approach is one step further in integrating efforts and aligning goals across all domains. All parties have commensurate rights in pursuing their own goals (e.g., performance for application owners, and revenue for edge providers), in the mean time being obligated to satisfying requirements and constraints posed by each other. Their awareness of each other, willingness to collaborate and form agreements, and means of enabling such collaboration are thus of crucial importance to success of this approach. These do not mean full disclosure of each party's intentions and information to the other parties (e.g., full details of the application software or resources in the edge). Each party can participate by only exposing the level of information that it is willing to disclose, for instance, a realization plan might only contain number and types of VMs/containers, guaranteed data rate or latency between VMs/containers, and associated cost; details of how these resources are orchestrated can be hidden from the application owner for security or privacy reasons. Technically, SOA or AAN allows one party to centrally manage the optimization task by taking inputs from other parties, while the co-design approach enables decentralized negotiation and optimization among all parties in a cooperative and automated manner.

B. Challenges and Future Directions

We have identified several challenges and places where efforts are needed for the co-design approach to be practical.

Federated optimization: The term *federated optimization* extends from federated learning, and is a generic method for distributed optimization while preserving data privacy of involved parties [12]. It allows the application owner and edge providers to jointly optimize for resource provisioning and negotiate the SLA terms, without revealing sensitive information such as network topology or application demands to each other. The method involves formulating a decomposable formulation, where each party holds part of the variables, constraints, and/or objective function. Parties are involved in a distributed iterative process such as primal-dual decomposition or distributed stochastic gradient descent to jointly solve the formulation. Messages exchanged only reveal intermediate results such as gradients or aggregate dual prices but no raw data of each party. Designing privacy-preserving federated optimization algorithms will not only address the privacy concerns, but also reduce the cognitive and operational overhead for secure data management in the co-design process.

Robust, risk-aware and adaptive optimization: Resilience is crucial at the edge [13], [14]. Robustness methods *proactively* tackle unexpected situations such as hardware/software faults or demand bursts, making sure that resources are always available to cover up all except some rare cases. Risk modeling and optimization further allow minimizing the probability and costs of potential SLA violations in the rare cases. Automated adaptation ensures that even a rare unfavorable situation arises, the system has contingency plans to ensure safe and acceptable operations until the situation is resolved.

Game theory-based incentives: Keep in mind that providers have goals centered around *revenue*, and application owners have budget considerations. Incentive mechanism, including pricing, cost management and violation penalty/compensation, is an integral part of an edge system [15]. An on-demand market mechanism, i.e., one where resource prices are closely tied to demand and supply of resources, works favorably toward everyone's interest: 1) application owners can purchase resources and SLAs only when needed without long-term financial commitment; 2) edge providers can set low prices at low demand to attract more buyers, and set high prices when demand is high to boost revenue. Mechanism design should however be strategy-proof and trustworthy. In other words, no one should neither be able nor has incentive to deviate from the aligned goal of all parties or deliberately harm the system. Truthful mechanisms, for instance, are valuable tools, which ensure that each party achieves the highest utility when behaving cooperatively.

Accountable design and SLAs: Traditional network measurement tools face difficulties in obtaining both end-to-end and per-component breakdown measurements in a trustworthy way. Distributed tracing with high-accuracy network-wide clock synchronization is a promising way. For all parties to trust the measurement result, the measurement process should be totally transparent or verifiable to every party involved. Note that this *almost* contradicts the privacy requirements of different parties, and hence privacy-preserving validation methods are needed. Promising methods include: trusted execution environments (TEEs) for providing trusted and verifiable results, privacy-preserving or secure computation for privacy guarantees, and/or blockchain or other technologies for distributed consensus between all parties.

VI. Summary

Edge computing is a key enabling technology for next-generation applications. However, reaping its full benefits is not trivial. Best-effort services are no longer suitable for performance-stringent applications like ADV or VR/AR. And yet end-to-end performance guarantee is not achievable without close cooperation across computing, networking and application domains. Application-network co-design represents a conceptual framework as well as a principled approach for realizing such cooperation. Core principles include agreement and accountability that are desired in any type of machine or human cohort, and abstraction and automation that are specific to the digitized edge computing ecosystem. Prerequisite technologies for the co-design approach are mostly ready, except pending design and implementation shifts mostly at the application side. The approach is well positioned in use cases such as ADV and multi-user VR/AR, but can be essentially applied in any application scenario that benefits from edge performance guarantees. Still, research and development are needed in regards to theoretical understanding and systems implementation of this approach in real life scenarios.

Acknowledgement

This work was supported in part by NSF grants 2007391, 2007469 and 2045539. The information reported here does not reflect the position or the policy of the funding agency.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, oct 2016.

[2] R. Yu, V. T. Kilari, G. Xue, and D. Yang, "Load Balancing for Interdependent IoT Microservices," in *Proc. IEEE INFOCOM*, 2019, pp. 298–306.

[3] R. Weingärtner, G. B. Bräscher, and C. B. Westphall, "Cloud Resource Management: A Survey on Forecasting and Profiling Models," *Journal of Network and Computer Applications*, vol. 47, pp. 99–106, jan 2015.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, jul 2014.

[5] D. Johnson, D. Maas, and J. Van Der Merwe, "Open Source RAN Slicing on POWDER," in *ACM MobiSys Demos*, 2021, pp. 507–508.

[6] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, P. Ruth, and E. Deelman, "FABRIC: A National-Scale Programmable Experimental Network Infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, nov 2019.

[7] B. Li, X. Peng, Q. Xiang, H. Wang, T. Xie, J. Sun, and X. Liu, "Enjoy Your Observability: An Industrial Survey of Microservice Tracing and Analysis," *Empirical Software Engineering*, vol. 27, no. 1, p. 25, jan 2022.

[8] R. Molina-Masegosa, J. Gozalvez, and M. Sepulcre, "Comparison of IEEE 802.11p and LTE-V2X: An Evaluation With Periodic and Aperiodic Messages of Constant and Variable Size," *IEEE Access*, vol. 8, pp. 121 526–121 548, 2020.

[9] ETSI TS 122 186, "5G; Service Requirements for Enhanced V2X Scenarios," Tech. Rep., 2020.

[10] W. Zhang, J. Chen, Y. Zhang, and D. Raychaudhuri, "Towards Efficient Edge Cloud Augmentation for Virtual Reality MMOGs," in *Proc. ACM/IEEE SEC*, 2017, pp. 1–14.

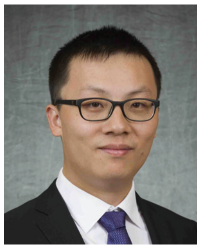
[11] S. Zhao and D. Medhi, "Application-Aware Network Design for Hadoop MapReduce Optimization Using Software-Defined Networking," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 804–816, dec 2017.

[12] J. Konečný, B. McMahan, and D. Ramage, "Federated Optimization: Distributed Optimization Beyond the Datacenter," pp. 1–5, 2015, date accessed: 2022-01-30. [Online]. Available: http://arxiv.org/abs/1511.03575

[13] V. Prokhorenko and M. Ali Babar, "Architectural Resilience in Cloud, Fog and Edge Systems: A Survey," *IEEE Access*, vol. 8, pp. 28 078–28 095, 2020.

[14] Y. Chen, B. Ai, Y. Niu, H. Zhang, and Z. Han, "Energy-Constrained Computation Offloading in Space-Air-Ground Integrated Networks Using Distributionally Robust Optimization," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 11, pp. 12 113–12 125, nov 2021.

[15] J. Moura and D. Hutchison, "Game Theory for Multi-Access Edge Computing: Survey, Use Cases, and Future Trends," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 260–288, 2019.



Ruozhou Yu (Student Member 2013, Member 2019, Senior Member 2021) is an Assistant Professor of Computer Science at North Carolina State University, USA. He received his PhD degree (2019) in Computer Science from Arizona State University, USA. His research interests include internet-of-things, cloud/edge computing, smart networking, algorithms and optimization, distributed machine learning, security and privacy, blockchain, etc. He has served on the organizing committees of IEEE INFOCOM 2022-2023 and IEEE IPCCC 2020-2023, and as members of the technical committee of IEEE INFOCOM 2020-2023. He is a recipient of the NSF CAREER Award in 2021.

Guoliang Xue (Member 1996, Senior Member 1999, Fellow 2011) is a Professor of Computer Science in the School of Computing and Augmented Intelligence at Arizona State University. His research interests span the areas of Internet-of-things, cloud/edge/quantum computing and networking, crowdsourcing and truth discovery, QoS provisioning and network optimization, security and privacy, optimization and machine learning. He received the IEEE Communications Society William R. Bennett Prize in 2019. He is an Associate Editor of IEEE Transactions on Mobile Computing, as well as a member of the Steering Committee of this journal. He served on the editorial boards of IEEE/ACM Transactions on Networking and IEEE Network Magazine, as well as the Area Editor of IEEE Transactions on Wireless Communications, overseeing 13 editors in the Wireless Networking area. He has served as VP-Conferences of the IEEE Communications Society. He is the Steering Committee Chair of IEEE INFOCOM.