Elevating the Edge to Be a Peer of the Cloud

Umakishore Ramachandran, Harshit Gupta, Adam Hall, Enrique Saurez, Zhuangdi Xu
Georgia Institute of Technology
Atlanta, Georgia
{rama,hgupta,ach,esaurez,xzdandy}@gatech.edu

***Abstract*—Enabling next generation technologies such as self-driving cars or smart cities requires us to rethink the way we support their applications. The emergence of these technologies is fueled by the proliferation of a large number of devices in the Internet of Things. These devices have the potential to generate massive amounts of data, and applications supporting them often require this data to be processed in a timely manner. Because of these requirements, we must augment and extend the Cloud computing model to better serve such applications. The backhaul links connecting clients to Cloud data centers could quickly become overwhelmed by such data, and the physical distance of these data centers from clients prevents low-latency response times. To meet the challenges posed by emerging IoT applications, we must provide Cloud-like functionality closer to the edge of the network, where clients and their data live. We propose to elevate the Edge to be a peer of the Cloud for addressing these challenges.**

***Index Terms*—Edge/Cloud, Distributed Systems, Scalability, Geo-distribution, Programming Idioms, Distributed Data, Execution Models, Orchestration**

I. Introduction

The confluence of advances in hardware devices and software applications serves as a driver for enabling next-generation technologies. As hardware capabilities increase, applications emerge and evolve to exploit those capabilities and realize new functionalities. A leading example of this phenomenon can be seen in the Cloud computing revolution, which commoditized hardware to support applications serving vast numbers of users from a few large, centralized data centers. Recently, this paradigm has reached an inflection point. The Internet of Things (IoT) has introduced an unprecedented number of low-cost hardware devices that constantly sense and generate data. At the same time, applications have emerged to convert this sensed data into actionable knowledge. Meeting all of the requirements of these applications from the Cloud alone is challenging for several reasons. First, the amount of data generated by devices can quickly saturate the bandwidth of backhaul links to the Cloud. Second, many applications require low-latency responses for making decisions on sensed data, which becomes difficult to achieve the further these devices are from Cloud data centers. And finally, there may be regulatory or privacy restrictions on the data generated by devices, meaning that such data should be kept in the same place where it is generated. For these reasons, enabling next-generation technologies requires us to reconsider the current trend of serving applications from the Cloud alone.

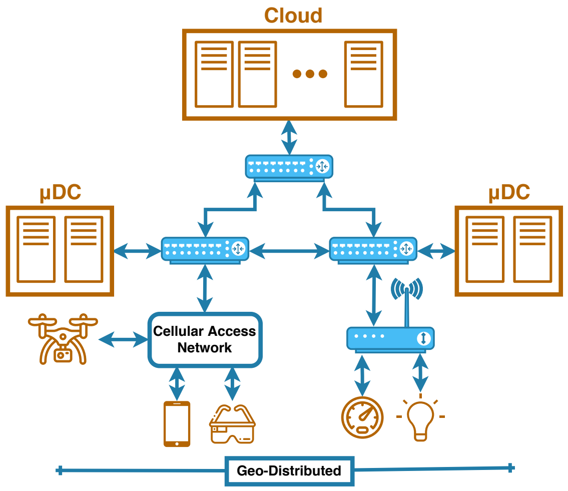

Fig. 1: An Edge computing ecosystem. IoT devices connect to micro data centers located across the edge of the network. Edge nodes in these data centers provide first stage data aggregation and processing, and ensure only relevant data is sent to the Cloud.

Over the past few years we have seen a growing interest in the use of Edge computing as a way to meet the unique challenges posed by emerging IoT applications. In the Edge computing paradigm, applications are served from a series of micro data centers that are geographically distributed throughout the last mile of the network. These micro data centers can be composed of hardware ranging from a few single-board computers to a couple racks of servers. By collocating computing resources close to client devices, we are able to provide several points of data aggregation with low-latency connectivity and localized processing and storage of data. The advantages of Edge computing align with the requirements of emerging IoT applications. Figure 1 depicts a geo-distributed Edge ecosystem. On the right, WiFi-connected IoT devices are served by a micro data center via a router with wired Internet access. On the left, *Augmented Reality (AR)* enabled mobile devices connect to the micro data center via the cellular network. Both of these micro data centers aggregate and pre-process data from all client devices and send relevant portions of this data to the Cloud for further processing.

The micro data centers that make up an Edge computing environment may exist in a variety of places, such as the wiring closet in a hotel or the base of a cell tower. Given the limited physical footprint of such environments, the hardware resources available in a micro data center are considerably less than those of its Cloud-based counterpart. Despite this limitation, we must provide a high degree of multi-tenancy at the Edge to support a variety of applications that may co-exist on the same IoT devices. Further, we must provide Cloud-like functionality such that developers need not struggle through creating highly distributed applications to run on a heterogeneous, widespread infrastructure. Enabling a seamless transition from hosting applications solely in Cloud data centers to micro data centers at the Edge means we must elevate the Edge to be a peer to the Cloud.

There are several challenges to extending the Cloud model to the Edge. These challenges can be grouped into four key research areas:

Programming Models: We must devise programming frameworks for facilitating the composition of complex latency sensitive applications across the Edge. These frameworks should allow developers to seamlessly adapt best practices from application development for the Cloud to the Edge.

Storage: We must create new geo-distributed data replication and consistency models commensurate with the network heterogeneity of the Edge while being resilient to coordinated failures. These models will allow us to provide a similar degree of data retention and retrieval as the Cloud on much less reliable infrastructures.

Autonomous Execution & Orchestration: Edge environments must provide support for the rapid dynamic deployment of application components to achieve multi-tenancy and elasticity with respect to limited computational, networking, and storage resources.

Runtimes: To achieve a high degree of multi-tenancy on limited hardware resources, we must look beyond traditional models of hosting applications in dedicated virtual machines or containers. Instead, we must develop efficient runtimes to allow a large number of applications to co-exist on a platform with the same isolation and security guarantees provided by Cloud hosting.

In this paper we present our vision of the direction in which the Edge and Cloud are moving. Although space does not permit us to delve into every detail, we present key points as follows. We begin with the discussion of a next-generation application that exemplifies the need for Edge computing, and then describe how each research area meets this need. Section II describes our next-generation application: an Augmented Reality enabled parade experience with millions of attendees. Next we describe the application's need for Programming Models in Section III, Autonomous Execution and Orchestration in Section IV, Storage in Section V, and Runtimes in Section VI. Finally, we conclude with an overview of the Edge-enabled future which represents the next evolution of Cloud computing.

II. A Next Generation Application for the Edge

To motivate our vision of the necessity for Edge computing, we present a futuristic example application which describes the evolution of one of the Cloud's most popular use cases.

Social media is considered a killer application of the Cloud computing revolution. The advent of this technology has enabled people to share photos, video, and text with friends and family in different locations. The next generation of social media applications will be far more immersive. People will use phones and other AR devices to bridge their virtual interactions with the real world. These interactions will be bolstered by the rich streams of data from IoT devices, providing unprecedented experiences.

Consider a scenario where people gather for a parade, such as the annual Macy's Thanksgiving Day Parade in New York City. Events of this type can host millions of spectators and thousands of exhibitors over parade routes spanning many city miles [5]. In our futuristic scenario, each spectator has a camera-equipped AR device (such as a phone or tablet) that overlays rich data feeds on their views of the event. Each of these views is highly individualized, and may include information about parade floats, commentary from other spectators, video or text conversations with friends, and data gathered from sensors along the route. These devices give spectators access to parts of the parade only available through AR, and allow them to interact with parade elements in real time. Additionally, parade personnel, law enforcement, and paramedics utilize AR-enabled headsets to assist spectators through tasks such as locating children who have been separated from parents or ensuring the safety and security of attendees by automatically monitoring for any suspicious activities. Providing these experiences requires us to process a large amount of data while ensuring consistency and availability among devices existing local to the same area in which spectators and parade exhibitions exist.

Enabling the AR-equipped application in the above scenario presents several challenges. First, we must process a series of high resolution video and sensor data over a shared communications infrastructure. If we conservatively estimate the data feed from each AR-enabled device will require 10 Mbps of bandwidth [7], the large number of parade attendees could quickly saturate a 10 Gbps uplink to the Cloud. Second, this data must be processed in a timely manner, such that information associated with the object in view may be retrieved and overlayed on that object without perceptible delay or inconsistency to the user. If two users are looking at the same part of the parade route, it is important that they

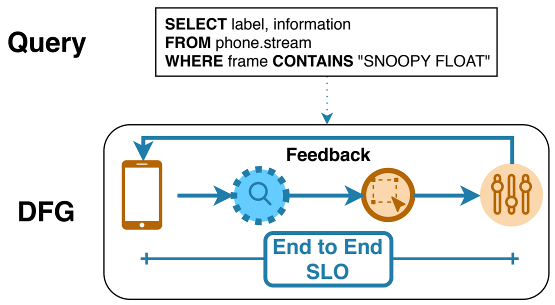
Query
SELECT label, information
FROM phone.stream
WHERE frame **CONTAINS** "SNOOPY FLOAT"
DFG
Feedback
End to End SLO

Fig. 2: An example application specified using a SQL-like declarative language and its corresponding DFG. It specifies an end-to-end latency SLO and contains the decomposed application with three stages: detection, filtering, and information extraction/feedback.

see the same common data displayed at the same time to enable a shared experience. This means video data must be transmitted with low latency. Chen et al. have shown processing data in the Cloud alone may introduce too much latency to achieve the required response times of interactive applications [8]. And finally, data associated with each object and user may be privacy sensitive in nature, meaning that data may need to remain local to its source. For example, an attendee could use their personal device to locate a family member in the crowd via facial recognition, and local laws may require this tracking to be performed within the same region where it originates. When we consider that similar operations are being performed in tandem by thousands of users within the same area, we quickly realize that enabling such functionality without the Edge can become untenable.

Supporting the requirements of a next-generation social media application across the Edge requires us to make inroads in several research areas. In the following sections, we discuss the challenges of each area and how they relate to the application scenario we just described.

III. Programming Models

More than a decade of research into Cloud computing has provided us with robust programming models that allow for the seamless development of applications for the Cloud. The reliability of the Cloud allows these models to abstract away the need to account for issues such as backend failures or latency and bandwidth limitations between services supporting application components. This is in contrast to development for the Edge, where creating effective applications currently requires several careful considerations. The placement and orchestration of application components at the Edge is difficult to perform manually, requiring domain expertise beyond the purview of most developers. At the same time, it is equally difficult to perform these operations in an automated fashion without some strong notion of application semantics. To address this problem, we must provide simple programming models that either implicitly or explicitly expose enough semantic information without creating undue burdens on developers.

The Edge-centric programming model that we provide must hide all the complexities of where and how a program should be executed to achieve optimal performance. For example, the programming model provided by the Apache Spark cluster-computing framework [4] allows developers to harness the power of big data without needing to understand the intricacies of placement and availability in a distributed computing environment. We must support the same degree of functionality for developers at the Edge by providing the following features in the programming model.

A. Composition & Synthesis of Application Pipelines

In our motivating scenario, users rely on video streams to enhance their experiences. One enhancement to this experience is the ability to identify and track parade objects. For example, a spectator may wish to be notified when the camera sees their favorite celebrity, and then retrieve information about the float on which the celebrity is riding. This operation requires performing image recognition on video frames from the AR device to identify the celebrity's current position and then tracking the celebrity and float across the spectator's field of view. A programming model at the Edge could provide a declarative framework with a SQL-like syntax to allow developers to express their intentions for tracking objects via image recognition without the need to understand the specifics of how the task is performed, similar to BlazeIT [13]. The declarative framework provides a way for users to intuitively compose queries for desired outcomes. The framework would automatically synthesize the query into a Data Flow Graph (DFG) where each node is a function that must be executed in a pipelined manner to service the query. In this way, domain experts in fields such as computer vision can guide the synthesis of complex application components without needing to be directly involved in the application process, and developers can benefit from this expertise without needing to learn new information unrelated to their discipline. Additionally, this decomposition helps the orchestrator to improve the resource utilization, which is discussed in more detail in section IV.

B. Performance Guarantees Through Service Level Objectives

From a developer's perspective, the unaided deployment of applications on an Edge computing infrastructure is a daunting task. Applications have diverse performance requirements, and to ensure these requirements are met the developer would need to keep track of variables such as the locations of both clients and micro data centers, the network connectivity statuses of these micro data centers, and the current resource capacities of nodes across the Edge. To alleviate this burden, developers should instead be provided with a way to express their application performance requirements as a series of *Service Level Objectives (SLOs)* which provide hints to the resource orchestrator (Section IV) about each application's requirements. The SLOs must specify end-to-end requirements so that all sources of performance overheads are considered during the orchestration of application components across a heterogeneous infrastructure (Section IV-A).

C. Failure Handling

When working within Edge computing environments, we cannot assume the availability of consistent power, cooling, or connectivity. As such, the framework should provide a strong failure handling model to ensure the pipelined execution of components in the DFG can recover and complete in the event of failures. For example, if one or more Edge nodes disappear during the pipelined execution of an image processing operation, the framework should recognize this event and redistribute workloads to available nodes while updating graph dependencies.

IV. Autonomous Execution & Orchestration

Applications expressed in the above programming model need to be executed on a heterogeneous Edge-computing infrastructure while respecting SLO specifications. Since resources in an Edge computing environment are relatively limited, ensuring they are carefully managed is essential to their efficient operation. These resources will exist across geo-distributed micro data centers with heterogeneous hardware configurations, so we cannot assume the same level of availability in network connectivity or some reliable central authority for governance. Instead, the control plane that oversees orchestration operations must be decentralized. This decentralized system will be responsible for monitoring the health and resource utilization of Edge nodes and using this information to make scheduling decisions. Several components are necessary to support such a system:

A. SLO-aware Deployment of Applications

Applications need to be executed on heterogeneous Edge computing infrastructures while adhering to end-to-end SLO specifications (Section III-B). The orchestrator should take specifications that have been synthesized by the programming model and use them to inform its decision on placing application components throughout the Edge. For example, there may be many applications that need to process video on a limited number of specialized GPU or ASIC hardware devices, meaning the orchestrator must carefully distribute workloads commensurate with the performance objectives of each application. In order to better estimate end-to-end performance, Edge-centric storage middleware and the programming model runtime should expose estimates of performance metrics like execution latencies to the orchestrator. In addition to these application-specific requirements, the orchestrator must also consider that power, cooling, and connectivity may be inconsistent across micro data centers, leading to a higher degree of entropy in the overall state of the resource pool.

B. Monitoring & Dynamic Reconfiguration

The Edge computing ecosystem is prone to a high degree of dynamism, which stems from circumstances such as the increase or decrease of activity in a specific geographic area, the mobility of clients, and differences in network connection quality due to congestion. For example, in our parade scenario some exhibits may be far more popular than others, leading to localized spikes in activity as those parts of the parade move through different points along the route. The orchestration engine must continually meet guaranteed SLOs despite this dynamism. A first step toward meeting this requirement is maintaining up-to-date knowledge on the state of the Edge computing ecosystem, for which continuous monitoring of that system is necessary. Resource monitoring is essential to tracking the current status of SLOs, but in a highly geo-distributed environment it can be complicated to scale and manage such monitoring operations. As a result, distributed monitoring implementations and aggregation are necessary. Monitoring the status of many applications across a vast infrastructure leads to a tremendous amount of data collection, which in itself is a source of high overhead. To enable decentralized monitoring at the Edge, efficient monitoring mechanisms must be created so that the responsiveness of the control plane is not slowed down by the overhead of observation. SLOs should be guaranteed in spite of system dynamism through the dynamic reconfiguration of applications. Such reconfiguration involves scaling application components both horizontally and vertically, as well as enabling the live migration of applications commensurate with client mobility.

C. Control Plane Decentralization

To achieve high availability and tolerance from unreliable networks, the control plane requires decentralized, largely autonomous components which can provide orchestration for different applications on the same shared infrastructure. Each of these orchestrator components should refer to its own knowledge of the infrastructure's current state to make decisions. In our parade example, an orchestrator which exists in the micro data center serving a block of the city would have immediate knowledge of its own environment and cached knowledge of the Edge computing infrastructure at large. Typically, orchestration algorithms assume that the state visible to them exists as the authoritative copy and is perfectly synchronized with the actual state of the infrastructure [19]. Depending on the structure of an application, state synchronization may be required if that application's components are placed across the domains of multiple controllers. To support this logic, state synchronization mechanisms must be built to allow decentralized orchestrator components to keep each other up-to-date, in turn providing the illusion of single-copy to orchestration algorithms.

D. Efficient Resource Utilization

Ensuring the orchestrator is able to maximize the efficient utilization of resources is of particular importance due to the limited hardware capacity of Edge nodes. There are three key components essential to this goal. First, the load generated by applications should be spread throughout a number of appropriate Edge nodes by use of geo-aware load balancing schemes. These schemes must understand the notion of correlated failure and the properties of geo-distributed hardware. Their interactions with the Programming and Storage frameworks can improve the quality of load balancing through context awareness. Second, the Programming Model's decomposition of applications into multiple components should be considered by the orchestrator. Individual application components from this decomposition may have their computation scheduled on different resources to improve utilization. As an added benefit, this decomposition can also allow the Edge environment to exploit the microservices trend that has been seen in Cloud environments. However, scheduling the computation of different application components can potentially require coordination between multiple micro data centers in close proximity to enable cross data center deployments. Finally, application components should be re-utilized whenever possible. To provide such functionality, these components must be discoverable and associated with some contextual information, since two instances of the same service may not perform the exact same operations. Additional labeling of application context and properties could help the orchestrator perform better matching operations, leading to more sound decisions in component reuse.

The orchestration framework is illustrated in Figure 3. In this diagram, distributed monitoring components track the health and performance of both micro data center hardware and application components and report these statistics to the distributed orchestrators. These orchestrators coordinate among each other to share monitoring information and use this knowledge to make intelligent management decisions guaranteeing end-to-end SLOs are met.

V. Storage

While the Programming Model and Control Plane we described handle the tasks of defining and scheduling application executions, we still must account for handling data generated by those applications. The generated data might contain an application's mutable state or might serve as input for another application. The typical way of handling such data is by leveraging platform services like key-value stores and publish-subscribe systems, so

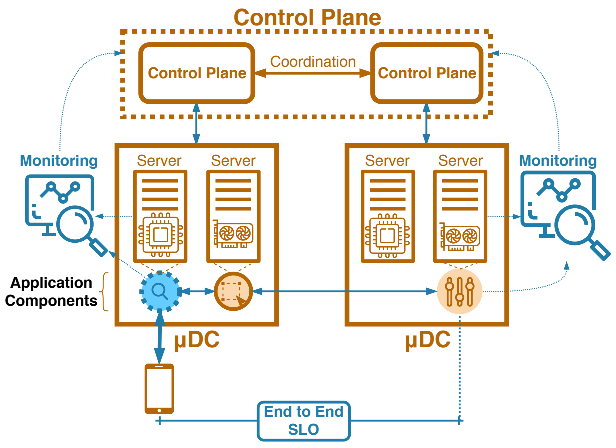

Fig. 3: An example orchestration framework, consisting of distributed monitoring and control operations to guarantee end-to-end SLOs.

that the management of data is abstracted away from the developer.

In a Cloud-only ecosystem, services can be provided by deploying off-the-shelf solutions like Apache Cassandra [16] and Pulsar [3] that are designed for reliable data center environments. Because of these design decisions, the divergent nature of an Edge computing infrastructure poses several challenges to such services. First, the inter-node latency in an Edge infrastructure is orders of magnitude higher than that of a Cloud data center. These services typically rely on consistently available connectivity with very low latencies to synchronize operations and data between instances. Second, an Edge infrastructure may be exposed to a variety environments, any of which may introduce conditions that can make the infrastructure more prone to failures. These failures can lead to split-brain scenarios where different instances are unable to perform the synchronizations required to maintain data consistency. Finally, the fact that multiple Edge computing nodes might share single sources of power and network connectivity makes correlated failures much more likely. Such failures increase the likelihood that data will be unavailable at the time a client requests it.

We elaborate on some of the approaches to addressing these challenges.

A. Data Partitioning for Low Latency

Data partitioning in key-value stores like Cassandra and publish-subscribe systems like Pulsar fundamentally relies on selecting the best server to host a particular *partition* of data. To ensure low-latency data access, the selection of the target server for hosting a data partition should be done by taking into account the *locations* of interested clients. One solution to enabling such functionality at the Edge comes from creating data store systems where a developer may provide constraints on data access latencies which the system uses to determine the optimal set of nodes for hosting each client's data to meet those constraints, as illustrated in Figure 4.

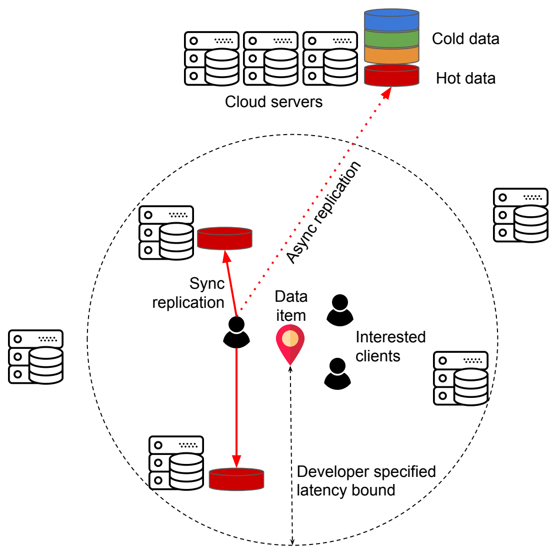

Fig. 4: A high-level view of storage between the Edge and Cloud. Relevant data is synchronized between nodes closest to clients who use it, and asynchronously replicated to the Cloud.

B. Consistency vs. Fault-Tolerance Tradeoff

Replication is the most popular approach to building fault-tolerant systems. However, with this approach comes the added expense of maintaining consistency between data replicas. To cope with such overhead, a number of solutions based on eventual consistency became prominent [6] [15]. However, researchers at Google observed that coping with eventual consistency takes up significant development time and introduces complications and bugs [18]. To provide strong consistency in a geo-distributed setting, all replicas should be located in proximity to each other, such that synchronization among them is fast. Unfortunately, doing so leads to an increased vulnerability to correlated failures [9]. One way to cope with this tradeoff is through the use of novel consistency models as introduced by Gupta, et al. [10] and Saurez at al. [17]. The consistency semantics in these works allow the system to perform geo-replication with a reasonable degree of fault-tolerance and lower overhead than traditional geo-replicated databases.

C. Interplay Between Edge and Cloud

The limited resource capacity on Edge nodes makes an Edge-Cloud interplay necessary. Since applications hosted at the Edge often have real-time requirements, the relevance of a data item is likely to decrease over time. In this respect, eventually each data item becomes irrelevant for real-time operations. Such older data-items could be evicted from Edge nodes and pushed to Cloud-based datastores for long-term retention and historical queries. Doing so conserves the limited storage resources of Edge nodes for critical/relevant data.

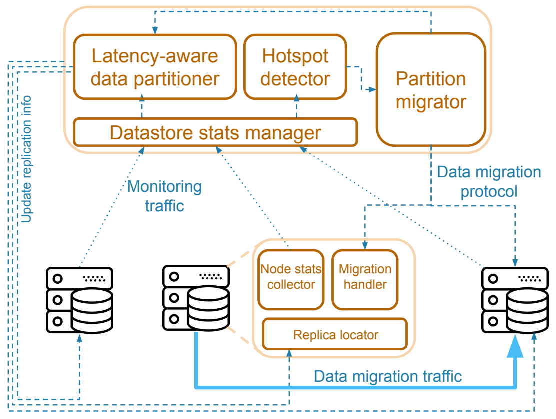

Fig. 5: Storage system control-plane architecture. Partitioning data and detecting workload hotspots is performed using performance metrics collected from individual nodes. Replica location metadata on each node is updated upon new partition creation or data migration.

D. Handling Skews in Workload

In a latency-based placement approach, data items often end up being stored on the nodes closest to the source of that data. In our motivating scenario, there are likely to be more people in popular sections of the parade route. This population distribution can lead to skews in activity distribution (e.g., a high density of spectators will generate far more data). Because of these skews, hotspots in workload distributions could develop which in turn would adversely impact the tail latency of responses [14]. One way to address this problem is through the use of data distribution schemes that are a hybrid of (1) schemes that partition data for low-latency, and (2) schemes that provide even load distribution. One such policy for key-value stores is proposed by Gupta, et al. [11], wherein the partition key of a data item is calculated using its location and a consistent hash of its timestamp and item-type field. Another approach worth considering is the live migration of data away from overloaded nodes in order to reduce the load on them and handle skews of a transient nature.

Each approach to addressing the challenges of storage at the Edge should be joined together in creating an overall storage architecture, as shown in Figure 5. This architecture would seamlessly provide the fast, reliable storage paradigms that developers are used to using in the Cloud, while accounting for the intricacies of Edge environments behind the scenes.

VI. Runtimes

In a traditional Cloud model, we may assume the notion of unlimited computing power. This assumption allows us to provision continuously running dedicated virtual machines or containers for hosting each application. In contrast, the limited computing resources available in a micro data center make it impractical to perform such provisioning in every situation. To enable the high degree of multi-tenancy needed to support a variety of applications running at the Edge, we must devise new paradigms for the runtimes which execute applications.

If we consider our motivating scenario, we can see that different application components will have different computational lifetimes. For example, if the social media platform is performing a large amount of image recognition processing while emergency personnel look for a missing child, it would make sense to devote a longer running container to these operations during periods of high activity. In contrast, if content generation from users is infrequent during the same period, it would make sense to serve these operations on an ad-hoc basis. These patterns suggest we can achieve a higher degree of efficiency by adopting a hybrid model that hosts applications commensurate with their needs.

One method for hosting applications on an ad-hoc basis comes from the notion of Serverless Computing, which is sometimes also known as Function-as-a-Service. In the Serverless computing paradigm, applications exist as single purpose functions which only execute for a limited period of time when they are called and then shut down until needed again. Each function is hosted in a separate container which is instantiated upon function invocation and destroyed after a brief period of inactivity. Since application components only exist on an as-needed basis, we avoid unnecessarily committing resources and in turn allow a greater degree of sharing for those resources. The Serverless computing model allows us to achieve the much higher degree of multi-tenancy needed to serve a large number of clients.

Longer running applications would best be served by the more traditional method of hosting in containers. Containers provide a lightweight mechanism for segmenting operating system components and limiting resource consumption specific to one or more processes. However, we still must ensure these containers are properly adapted for Edge computing scenarios. For example, situations may arise where long-running applications must be temporarily pre-empted to serve spikes in demand for short-term applications. Ensuring that we may safely make these trade-offs allows us to provide many of the benefits of a traditional hosting model when necessary while still maximizing the utilization of limited resources.

In both scenarios, opportunities for innovation exist in two ways. First, we can improve container runtimes to meet the requirements of the Edge. The runtimes in state-of-the-art container frameworks include a broad array of functionality which may not be needed in an Edge computing environment. Such unnecessary functionality adds overhead which manifests itself in issues such as the *cold start problem*, where container instantiations introduce delays of 300 ms or more before a function can begin execution. These delays destroy the low-latency advantages of hosting applications at the Edge and must be reduced or eliminated. The second opportunity for innovation exists in developing new runtime methods

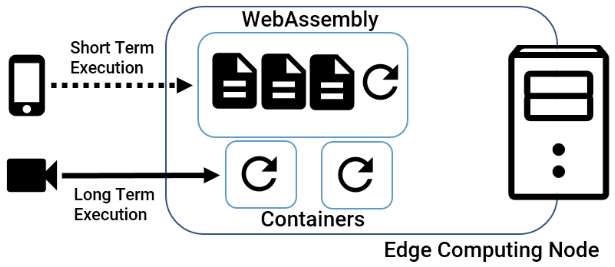

Fig. 6: A hybrid execution platform leveraging WebAssembly and Container runtimes. Short-term executions are handled by WebAssembly, whereas longer running applications are handled by Containers.

for executing applications. For example, Hall et al. [12] discuss the use of the *WebAssembly binary format* as a way to provide strong isolation and resource provisioning mechanisms to Serverless functions without the overhead of containers.

A hybrid runtime could leverage innovations from both approaches to provide low-latency application executions. For short-running applications, a lightweight model such as WebAssembly could be used in a Serverless platform to provide fast loading and portability. For longer running applications, the traditional container-based model could be used. The execution platform could switch between serving applications with either model based on their recent characteristics. This platform can be seen illustrated in Figure 6. In this diagram, short-term application executions from smartphones are handled by a single WebAssembly runtime. Hosting applications with WebAssembly allows for a high degree of efficiency since each application is executing as a thread under the same runtime process. Long running video processing applications from smart cameras execute in containers, with each container existing as one or more separate processes.

VII. Conclusion

The task of elevating the Edge to be a peer of the Cloud is not without challenges. However, it is important to meet these challenges if we wish to enable the broad set of applications fueled by the Internet of Things boom. These applications all share common requirements for dynamic scalability, low-latency communication, and efficient in-network processing to provide the *Sense-Process-Actuate* workflow used in dealing with real-world data streams, suggesting that there is a common solution to meet their needs.

Cloud computing has proven a good solution for serving applications at human perception speeds, many of which are throughput oriented web applications with humans in the loop. However, when we consider that many latency-sensitive IoT applications operate at computational perception speeds, we see that the centralization and physical distance of the Cloud from client devices makes it difficult to support the workflow of these applications. Often such applications operate independent of human intervention, relying on machine-in-the-loop processing to perform rapid decision making. To handle such workloads, we must provide a way to deal with their data close to the source. One solution to this problem comes from the Edge computing paradigm.

The notion of Edge computing seeks to extend the utility of the Cloud to the last mile of the network. Its goal is to provide this utility through resources that are hierarchical and geo-distributed in nature. Unfortunately, today the Edge exists as a slave of the Cloud. Platforms such as Azure IoT Edge [2] and Amazon AWS Lambda [1] provide Edge computing capabilities, but still rely on the Cloud for the core of their operations. To overcome this limitation, we need to make the Edge autonomous, such that it functions even when disconnected from the Cloud.

To enable an autonomous Edge, horizontal peer-to-peer interactions among Edge nodes is essential. This need stems from the fact that different interacting client devices may be connected to different Edge nodes at the same time. Such interactions are made possible by programming model, orchestration, storage, and runtime paradigms which are specifically tailored to the unique challenges of the Edge. By building these paradigms, we elevate the Edge to be a peer to the Cloud, in turn creating next evolution in Cloud computing.

References

[1] AWS Lambda. https://aws.amazon.com/lambda/. Accessed: 05-01-2019.

[2] Azure IoT Edge. https://azure.microsoft.com/en-us/services/iot-edge/. Accessed: 05-01-2019.

[3] Pulsar. https://pulsar.apache.org. Accessed: 05-01-2019.

[4] Spark Programming Guide. https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html. Accessed: 05-01-2019.

[5] The Macy's Thanksgiving Day Parade 2016 By The Numbers. https://www.forbes.com/sites/hayleycuccinello/2016/11/23/the-macys-thanksgiving-day-parade-2016-by-the-numbers. Accessed: 05-04-2019.

[6] Abadi, D. Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. *Computer 45*, 2 (2012), 37–42.

[7] Braud, T., Bijarbooneh, F. H., Chatzopoulos, D., and Hui, P. Future networking challenges: The case of mobile augmented reality. In *IEEE 37th International Conference on Distributed Computing Systems* (2017), pp. 1796–1807.

[8] Chen, Z., Hu, W., Wang, J., Zhao, S., Amos, B., Wu, G., Ha, K., Elgazzar, K., Pillai, P., Klatzky, R., et al. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (2017).

[9] Couto, R. S., Secci, S., Campista, M. E. M., and Costa, L. H. M. Latency versus survivability in geo-distributed data center design. In *IEEE Global Communications Conference* (2014), pp. 1102–1107.

[10] Gupta, H., and Ramachandran, U. Fogstore: A geo-distributed key-value store guaranteeing low latency for strongly consistent access. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems* (2018), ACM, pp. 148–159.

[11] Gupta, H., Xu, Z., and Ramachandran, U. Datafog: Towards a holistic data management platform for the iot age at the network edge. In *USENIX Workshop on Hot Topics in Edge Computing* (2018).

[12] Hall, A., and Ramachandran, U. An execution model for serverless functions at the edge. In *Proceedings of the International Conference on Internet of Things Design and Implementation* (2019), ACM, pp. 225–236.

[13] Kang, D., Bailis, P., and Zaharia, M. Challenges and opportunities in dnn-based video analytics: A demonstration of the blazeit video query engine. CIDR.

[14] Khare, S., Sun, H., Zhang, K., Gascon-Samson, J., Gokhale, A., and Koutsoukos, X. Ensuring low-latency and scalable data dissemination for smart-city applications. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation* (2018), pp. 283–284.

[15] Kirkell, J. Consistency or bust: Breaking a riak cluster, 2011.

[16] Lakshman, A., and Malik, P. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review 44*, 2 (2010), 35–40.

[17] Saurez, E., Balasubramanian, B., Schlichting, R., Tschaen, B., Huang, Z., Narayanan, S. P., and Ramachandran, U. METRIC: A Middleware for Entry Transactional Database Clustering at the Edge. In *Proceedings of the 3rd Workshop On Middleware for Edge Clouds & Cloudlets* (December 2018).

[18] Shute, J., Vingralek, R., Samwel, B., Handy, B., Whipkey, C., Rollins, E., Oancea, M., Littlefield, K., Menestrina, D., Ellner, S., et al. F1: A distributed sql database that scales. *Proceedings of the VLDB Endowment 6*, 11 (2013), 1068–1079.

[19] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems* (2015), ACM, p. 18.