Towards General Purpose Computations at the Edge

A preprint part of Book Chapter 2: Programming Models and Runtimes in Ultrascale Computing Systems*

Ali Shoker†, João Leitão‡, Peter Van Roy§, and Albert van der Linde¶

February 27, 2019

Originally designed to exploit the power of multi-core processors through virtualization, Cloud Computing [1] has changed over the past decade to support ultrascale computations. The new paradigm, often called *aggregation*, collects a large number of resources in a pool to form a single service with huge storage and computation capacities. Unfortunately, with the huge amounts of data generated via modern applications, the cloud center has become a bottleneck and a single point of failure. This advocated an extended paradigm, called *Edge Computing*, that brings part of the data storage and computation closer to the user. The benefits are plenty: reduced delays, high availability, low bandwidth usage, improved data privacy, etc. In this section, we introduce recent advances in edge computing that makes the coordination of edge networks synchronization-free and convergent. We address the main challenges facing applications on the data management and communication aspects. The section also provides convenient runtime environments for different categories of edge computing scenarios[1].

1 Motivation

Edge Computing offers the opportunity to build new and existing ultrascale applications that take advantage of a large and heterogeneous assortment of edge

*Book: `https://digital-library.theiet.org/content/books/10.1049/pbpc024e_ch2`

†HASLab, INESC TEC & University of Minho, Portugal

‡Universidade Nova de Lisboa, Portugal.

§Université Catholique de Louvain, Belgium.

¶Universidade Nova de Lisboa, Portugal.

[1]Credits go to all team members contributed to the success of this work within the EU FP7 Syncfree project and EU H2020 LightKone project. The research leading to these results has received funding from the European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 732505, LightKone project.

devices and environments. Fully realizing the opportunities that are created by edge computing, requires dealing with a set of key challenges related with the high number of different components that compose such systems and the interactions among them. In this work, was address the main challenges on the communication and data management levels allowing for robust communication and available data access.

On the communication frontend, the fact that applications are composed of components running in heterogeneous environments requires robust and efficient solutions for tracking these components. This implies the development of highly robust and adaptive membership services and mechanisms that allow efficient communication among these components. Among the promising class of gossip-based communication protocols are those "hybrid" ones [2, 3], in which payloads are propagated though an elected logical *spanning tree*, supported by lightweight meta-data across the graph for recovery (reconstructing another logical tree) under failures.

The consequences of such hostile environments are also present on the data management level. Since application components run on different administrative domains scattered across heterogeneous environments, communication links between these components can be disrupted by external factors (i.e, network partitions) frequently. This implies that the progress of computations executed across different application components cannot depend on continuous communication with other components, or in other words, cannot depend on synchronous interactions. This advocates the use of synchronization-free (i.e., sync-free) programming abstractions backed by sync-free data propagation and replication techniques. An interesting approach is to make use of Conflict-free Replicated Data Types (CRDTs) [4, 5, 6] that are proven abstractions designed to acheive convergence under such conditions (this is explained later in more details).

Finally, heterogeneity is the norm in ultrascale edge applications, and it exists at various layers: executiuon environments, communication media, data sources, operating systems, programming languages, etc. Addressing this heterogeneity can be achieved by leveraging on different run-time supports and frameworks that provide a more unified vision of resources to application developers. These different run-time and frameworks will have to inter-operate through the use of standard protocols and common data representation models.

In the following we refine the challenges associated with tapping on edge computing to design ultrascale applications, and discuss enabling technology that paves the way to tackle these challenges, and finally discuss a set of run-time and framework support that can simplify the design of such applications.

2 Edge Computing Opportunities

Edge Environments. To the contrary of cloud computing where the data and computation is centralized at the cloud data centers, the edge computing paradigm encompasses a large number of highly distinct execution environments that are defined by the network topology, connectivity, locality, and the storage

and computation capacities of the devices used. In particular, we identify we identify the following interesting edge environments:

- Fog Computing: a variant of cloud computing where the cloud is divided into smaller cloud infrastructures located in the user vicinity. In such environments, each fog cloud often serves as an individual cloud, although the data can eventually be incorporated with other fog clouds [7, 8].
- Mobile Cloudlets: small cloud datacenters that are located at the edge and are tailored to support mobile applications with powerful computations and low response times, e.g., in ISP gateways or 5G towers [9, 10, 11].
- Hardware-based Clouds: self-contained devices, such as routers, gateways, or set-up boxes, that are enriched with additional computational and storage capabilities like [12, 13].
- Peer-to-Peer (P2P) Clouds: these environments try to leverage existing devices, e.g., user mobiles, laptops, and computers in volunteer networks, aiming to cooperate towards achieving a common goal [14, 15, 2].
- Things and Sensor Network Clouds: resource constrained devices, e.g., Internet of Things devices, sensors, and actuators, capable of performing some computations on data without accessing or delegating to the (possibly unreachable) cloud center [16].

All of these different scenarios are characterized by having highly heterogeneous devices in terms of processing power and memory, but also regarding their connectivity to the backbone of the Internet or even their up-times (being continually running or being operating for only small periods of time). These different devices naturally, run different operating systems, from general purpose Linux based operating systems in the case of servers in cloud and private infrastructures, to proprietary operating systems in the case of set-up boxes, mobile operating systems, general purpose multi-user operating system or even single process operating systems in the case of small sensors and actuators. Gathering the capacity of devices with very different properties is highly challenging, and devising solutions that can exploit devices located in different edge devices brings additional challenges. Next we will discuss some of the key high level challenges in tapping the potential of the edge.

Challenges at the Edge Despite the diversity of edge computing environments, components, and properties, the major challenges are common to most of the scenarios. In particular, we recognize the following four challenges:

Scalability. One of the reasons to move the data and computation off the cloud data center to the edge is to reduce the I/O overload on the cloud and avoid bottlenecks related with the limited network capability connecting clients to the cloud infrastructures. Nevertheless, this raises another challenges on

handling the data and computation in a distributed way especially in ultra-scale systems composed of, potentially, many data centers and thousands of edge devices. This scale requires special techniques across the data, computation, and communication planes. As captured by the CAP theorem [17], and because scaling out will increase the potential for network portions, link failures, and arbitrary communication delays, ensuring availability—as an essential requirement for most applications including novel edge applications—requires relaxing the consistency model employed in the design and implementation of these solutions. Consequently, the computation should also be decentralized and coordinated to achieve the common goals of the entire system. Finally, the communication middlewares should also scale to afford a high number of nodes, e.g., through asynchronous, P2P, or gossip protocols.

Interoperability. Considering the edge categories discussed above, one can notice the notable diversity level of the devices and platforms used within the same or across edge clouds. This brings interoperability challenges if all components shall communicate with each others, thus requiring well studied interfaces and possibly introducing a common layer that all components can understand without compromising the characteristics deemed essential.

Resilience. While cloud datacenters use high quality equipment for the network and devices, edge computing often use commodity equipment that are far from perfect regarding failures. The problem is extrapolated with edge network problems that are likely to be loosely connected, mobile, and hostile. This threatens the quality of the service and makes the data and communication components even more complex. That said, one must consider the performance as well as the cost trade-offs (being a major factor due to the constrained resources).

Security and Privacy. Given the heterogeneity of the edge applications, security and privacy measures must be analyzed and tackled individually. However, in general, it is desired to find a common security layer or security measures that govern a wide range of applications. Security and privacy on the edge need to be addressed on the infrastructure and data levels. The former can be deployed at the communication or network layer, ranging from establishing secure connections to enforcing secure group dynamics, and cover several dimensions including data integrity, data privacy, or resilience to DoS attacks. On the other hand, edge applications often deal with sensitive data which likely requires lightweight encryption and data sanitization techniques to control the disclosure of such data. These may also include secret-sharing, anonymization, noise addition or partitioning, etc., depending on the specific security and functional requirements of the implementations.

Use Cases. As discussed in the edge environments, edge computing supports a plenty of applications and use-cases. In this section, we focus on three categories in which most of the use-cases lie:

- Time series applications. This category spans a multitude of applications

with the popularity of IoT. The scenario is often a type of time series where data is generated by the IoT devices, e.g., sensors, and pushed to the edge devices to get stored, aggregated, and partially computed. The aggregated data is then pushed to the center of the cloud for further handling. The data-flow can sometimes be in the opposite sense if actuator devices exist; in this case, the processed data in the cloud is pushed back to the actuators to do some action. Consequently, this scenario represents a hybrid model of light and heavy devices, different types of networks (e.g., Zig-bee, WIFI, WAN, etc), as well as data-flow direction.

- Mobile edge applications. This category covers all the applications in which devices are mobile and public. This makes the model very hostile as link failure and delays are expected, and the availability of nodes cannot be guaranteed (e.g., a mobile device can be switched off). The communication in such use cases does not follow a particular data-flow pattern, but it is often P2P or gossip-based due to the dominant dynamic graph-like network of nodes. In such applications, devices have moderate storage and computation resources that makes the interaction symmetric. Obviously, the main challenges in such use-cases are reseliance and availability. In some cases, access points, towers, or routers with more capacities can assist in storage, computation, and communication, which can be used as third party authority when needed.

- Highly available databases. This category is a natural evolution of scalable databases in cloud and cluster systems. The intuition is to replicate the database geographically, brining replicas or cache servers closer to the user. In this scenario, devices are at least commodity computers or servers with non-scarce capacities, and then network is often the Internet. In addition to availability, the challenge in such use-cases is to tolerate network partitions and optimize data locality (especially when partial replication is used). These scenarios are close to Fog Computing and Cloudlets with the difference that all node must work as a single (often loosely) coordinated system.

3 Enabling Technologies for the Edge

Synchronization-Free Computing. Edge devices and edge networks are both unreliable. This follows both from their design, e.g., they are low-power systems that are often offline, and from the nature of the edge itself, e.g., it is directly involved with real world activities, such as in Internet of Things. Despite this unreliability, we would like to perform computations directly on the edge.

To perform computations directly on the edge, we need distributed data structures and operations that tolerate the unreliability of the edge. Synchronization-free computing fits the bill because of its very weak synchronization requirement. A prominent example is Conflict-free Replicated DataType (CRDT), which is a

replicated data type that is designed to support temporary divergence at each replica, while guaranteeing that when all updates are delivered to all replicas of a given instance, they will converge to the same state. (More details about CRDTs can be found in Chapter 4 or by referring to [4, 5, 6].) CRDTs naturally tolerate node problems, namely nodes going offline and online and node crashes, and network problems, namely partitions, message loss, message reordering, and message duplication. Node crashes are tolerated as long as the desired state exists on at least one correct node. The following results on CRDT computations are summarized from [18].

CRDT Definition. For the purposes of this section, we define a *CRDT instance* to be a replicated object that satisfies the following conditions:

- Basic structure: It consists of n replicas where each replica has an initial state, a current state, and two methods, query and update, that each executes at a single replica.
- Eventual delivery: An update delivered at some correct replica is eventually delivered at all correct replicas.
- Termination: All method executions terminate.
- Strong Eventual Consistency (SEC): All correct replicas that have delivered the same updates have equal state.

This definition is slightly more general than the one given in the original report on CRDTs [4]. In that report, an additional condition is added: that each replica will always eventually send its state to each other replica, where it is merged using a join operation. This condition is too strong for CRDT composition, since it no longer holds for a system containing more than one CRDT instance. We explain the conditions needed for CRDT composition in the next section.

CRDT Composition. The properties of CRDTs make them desirable for computation in distributed systems. It is possible to extend these properties to full programs where the nodes are CRDTs and the edges are monotonic functions. To achieve this, it is sufficient to add the following two conditions on the merge schedule, i.e., the sequence of allowed replica-to-replica communications:

- Weak synchronization: For any execution of a CRDT instance, it is always true that eventually every replica will successfully send a message to each other replica.
- Determinism: Given two executions of a CRDT instance with the same set of updates but a different merge schedule, then replicas that have delivered the same updates in the two executions have equal state.

The first condition allows each CRDT instance to send the merge messages it requires to satisfy the CRDT conditions. The second condition ensures that the execution of each CRDT instance is deterministic, which makes it a form of

functional programming. We remark that SEC by itself is not enough for this, since the states of replicas *in different executions* that have delivered the same updates can be different, even though SEC guarantees that they are equal in the same execution. In practice, enforcing determinism is not difficult but it depends on the type of the CRDT instance. Article [18] explains how to do it for a set that has add and remove operations (the so-called Observed-Remove Set).

We define a *CRDT composition* to be a directed acyclic graph where each node is a CRDT instance, and each node with at least one incoming edge is associated to a function of all incoming edges arranged in a particular order. Given the first of the two conditions introduced above, we can show that the execution of a CRDT composition satisfies the same properties as a single CRDT instance. If the second condition is added, then the CRDT composition behaves like a functional program.

Hybrid Gossip Communication. Gossip is a well known and effective approach for implementing robust and efficient communication strategies on highly dynamic and large-scale system [15, 3]. In its most simple form, in a gossip protocol, each node periodically interacts with a randomly selected node. In this interaction both exchange information about their local state (and potentially merge it). Since all nodes do this in parallel and in an independent fashion, after approximately one round-trip time, all nodes will have performed, at least, one merge step, and on average two merge steps (one initiated by the node itself and another initiated by some peer). We usually call this period of interactions a *cycle*. After a small number of cycles, the network converges to a globally consistent vision of the system state. This simple approach cab be used, for instance to compute aggregate functions, such as inferring the network size or load. Interestingly, this can also be used for other, and more complex, purposes such as managing the membership of large-scale system, which implies building and maintaining an overlay (i.e, logical) network topology, in a way that is both robust and scalable, but also to support robust data dissemination in such systems.

Gossip-based approaches have been shown to be highly resilient to network faults, due to the inherent redundancy that its core to the design of gossip protocols. Unfortunately, this redundancy also leads to efficiency penalties. Hybrid gossip addresses this aspect of gossip protocols. In a nutshell, the key idea of hybrid gossip is to leverage on the feedback produced by previous gossip interactions among nodes, such that an effective and non-redundant structure of communication can naturally emerge. The topology of this *emergent structure* depends on the computation being performed by nodes, and it enables nodes significantly improve the communication and coordination cost by restricting the exchange of information among node to the logical links that belong to this structure, lowering the among of redundant communication.

Key to maintaining the fault-tolerance of gossip protocols in hybrid gossip is the use of the remaining communication paths among nodes (those that are

not selected to be part of the emergent structure) to convey minimal control information. This control information enables the system to detect (and recover) from failures that might affect the emergent structure. Moreover, in highly dynamic scenarios, the additional communication paths allow nodes to fall back to a pure gossip strategy, for instance, when there are a significant number of concurrent nodes crashes or network failures.

Interesting, hybrid gossip solutions naturally allow different components of the system to operate using either the emergent structure or a pure gossip approach simultaneously. Hence, components of the system that are in stable conditions (i.e, low membership dynamics and low failures) will operate resorting to the emergent structure, while components of the system that are subjected to high churn or network/node failure will fallback to use pure gossip while still being able to inter-operate with the components using the emergent structure.

Therefore, hybrid gossip approaches enable applications to, effectively and transparently, benefit from the resilience of a pure gossip approach entwined with the efficiency of a gossip approach that leverages an emergent communication topology. The hybrid gossip approach has been introduced in [2, 19]. The Plumtree protocol in particular, shows how to build an efficient and robust spanning tree connecting large number of nodes to support reliable application-level broadcast. This solution is currently used in industry, for example, the Basho Riak database uses it to manage the underlying structure of its ring topology which is used to map data object keys into nodes (through consistent-hashing).

4 Runtime for Edge Scenarios

Above we have discussed enabling technologies that can be leveraged to build new and exciting edge applications in the ultrascale domain. Tapping into these enabling technologies can however, be a complex task for developers. Therefore, it becomes relevant to provide frameworks, tools, and other artifacts that exploit these technologies in a coherent way, providing high level abstractions to programmers that aim at developing their ultrascale edge applications. We now discuss some existing runtime support tools and frameworks that have been recently proposed to this end.

Antidote. Antidote is a geo-replicated key-value store, designed for providing strong guarantees to applications while exhibiting high availability, thus providing a good compromise in the consistency versus availability trade-off in the design of cloud databases. These proprties make Antidote a strong candidate as an edge database especially when edge nodes have non-scarce resources (e.g., commodity servers).

In particular, some cloud databases adopt a strong consistency model by enforcing a serialization in the execution of operation, leading to high latency and unavailability under failures and network partitions. Other databases adopt a weak consistency model where any replica can execute any operation, with updates being propagated asynchronously to other replicas. This approach leads to low latency and high availability even under network partition, but replicas

can diverge. On the other hand, Antidote allows any operation to execute in any replica, but provides additional guarantees to the application as we explain next.

First, Antidote relies on CRDTs for guaranteeing that concurrent updates are merged in a deterministic way. Antidote provides a library of CRDTs with different concurrency semantics, including registers, counters, sets and maps. The applications programmer must select the most appropriate CRDT, considering its functionality and concurrency semantics (e.g., add-wins, remove-wins).

Second, Antidote enforces causal consistency, guaranteeing that whenever an update u may depend on update v, if a client observes update u he also observes update v. Applications can leverage this property to guarantee their correctness when the correctness depends on the order of updates, e.g., an update executed after changing the access control policies should not be visible in a replica with the old access control policies.

Third, Antidote provides a highly available form of transactions, where reads observe a causally-consistent snapshot of the database and writes are made visible atomically. Unlike standard transactions, write-write conflicts are solved by merging the concurrent update. Applications can leverage these highly-available transactions to guarantee that a set of updates is made visible atomically.

Fourth, Antidote provides support for efficiently enforcing numeric invariants, such as guaranteeing that the value of a counter remains larger than 0. To this end, it includes an implementation of a Bounded Counter CRDT [20], a shared integer that must remain within some bounds. The implementation uses escrow techniques [21] for allowing an operation to execute in a replica without coordination in most cases.

Finally, associated with Antidote, we have developed a set of tools to verify whether an application can execute correctly under weak consistency, and when this is not the case, what coordination is necessary. These tools are backed by a principled approach to reason about the consistency of distributed systems [22].

Antidote is designed to be deployed in a set of geo-distributed data centers. Within each cluster, data is sharded among the servers. Data is geo-replicated across data centers. The execution of transactions in Antidote, and the replication of updates across data centers, is controlled by Cure [23], a highly scalable protocol that enforces transactional causal+ consistency (combining CRDTs for eventual consistency, causal consistency and highly available transactions).

Legion. Legion [24] is a new framework for developing collaborative web applications that transparently leverage on the principles of edge computing by enabling direct browser-to-browser communication. Legion was implemented in *javascript* and it uses the *Web Real-Time Communications* (`https://webrtc.org`) to establish direct communication channels among web application users. At its core, Legion enables applications to transparently replicate, in the form of CRDTs, relevant application state in clients. Clients can then modify the application state locally, and through the use of hybrid gossip mechanisms, synchronize directly among them, without the need to go through the web application server. The server however is still used both to ensure the durability of the application state, but also to assist in the operation of Legion, namely

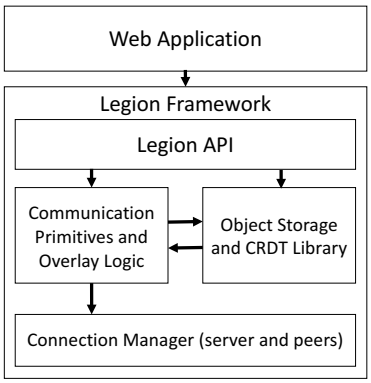

Figure 1: The Legion architecture (adapted from [24])

to simplify the task of creating the initial webRTC connections among clients when they enter the application.

A simplified architecture of Legion is illustrated in Figure 1. Legion can be used by a web application simply by importing a javascript script. This script provides the application access to the *Legion API*. The API exposes to the application the ability to manipulate data objects that can be used to model the application state. These data objects include records, counters, lists, and maps. All of these objects are internally represented by Legion through CRDTs which simplifies the the direct synchronization among clients of shared application state. This is provided by an extensible CRDT Library that is part of the *Object Store* component of Legion. The synchronization of objects among clients (and that of a subset of clients with the server to ensure durability) is transparently managed by the Object Store.

To guide the synchronization process, Legion leverages on an unstructured overlay network, whose construction is guided by the principles of hybrid gossip, and takes into consideration the relative distance of each client among them. This allows clients to mostly interact and synchronize with clients that are in their vicinity. While the typical use case in Legion is to have clients interacting through the manipulation of shared data objects, web applications also have access to communication primitives that enable them to disseminate messages among the currently active clients of the application in a decentralized fashion. This is achieved by a gossip-based broadcast protocol that operates on top of the legion overlay network.

Finally, Legion also takes into account security, by ensuring that before clients can start to replicate and manipulate application data objects they authenticate on a server. Moreover, Legion exposes an adapter API, that allows developers to integrate their Legion-backed applications with existing backends. The framework provides adapters to the Google Real Time API[2]. These adapters allow the developers to leverage this backed to do any combination of the fol-

[2]https://developers.google.com/google-apps/realtime/application

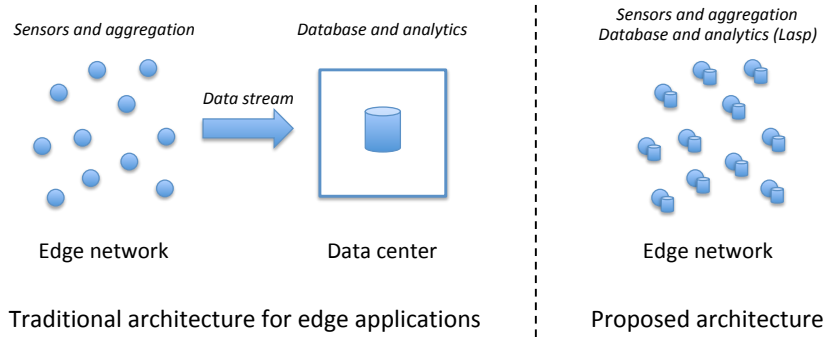

Figure 2: Proposed architecture for edge applications using Lasp

lowing: authentication and access control, data storage for durability, and support to the WebRTC signaling protocol required to create webRTC connections among browsers. More details on the design and operation of Legion can be found in [24]. Legion is open source and available, along side some demo applications through `https://legion.di.fct.unl.pt`.

Lasp. The Lasp language and programming system [25] was designed for application development on unreliable distributed systems, and in particular for edge computing. Lasp allows developers to write applications by composing CRDTs, as explained above [18]. In addition to composition, Lasp also provides a monotonic conditional operation that allows executing application logic based on monotonic conditions on CRDTs. The Lasp implementation combines a programming layer based on synchronization-free computing with a communication layer based on hybrid gossip. This makes the implementation highly resilient and well-adapted to edge networks.

Many of today's edge applications use the cloud as a database to store data coming from the edge. By using Lasp as their database, such applications can be translated to fully run on the edge (see Figure 2). This cannot be done with traditional cloud databases since they are not designed to run on unreliable edge networks. In the proposed architecture, the edge network runs everything: the sensors and aggregation software on individual edge nodes, and the database (Lasp) on all edge nodes. Analytics computations can be run either as an internal Lasp computation or external to Lasp on individual nodes, using Lasp just as a database.

Example Lasp program. A typical application for Lasp is the scenario of advertisements counter that counts the total number of times each advertisement is displayed on all client mobile phones, up to a preset threshold for each. Figure 3 defines graphically part of the Lasp program for this application. The actual code is a straightforward translation of this graph. The application has the following properties:

- Replicated data: Data is fully replicated to every client in the system. This replicated data is under high contention by each client.

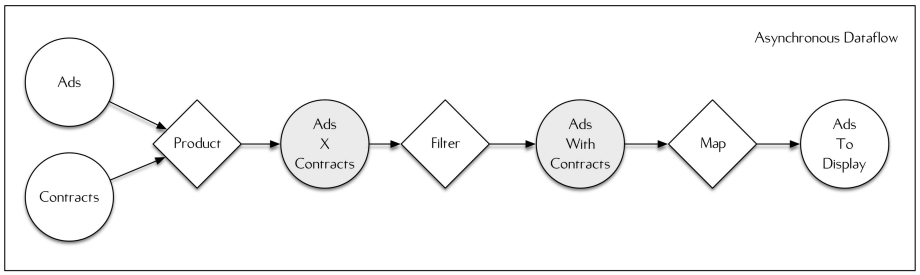

Figure 3: A Lasp computation to derive the set of displayable advertisements in the advertisement counter scenario. On the left, Ads and Contracts give information for the advertisements, including how many times they have been displayed, and their contracts, including the threshold for each advertisement. On the right are the advertisements that can be displayed. All data structures are sets, similar to database relations, and the computation is similar to an incremental SQL query.

- High scalability: Clients are individual mobile phone instances of the application, thus the application should scale to millions of clients.
- High availability: Clients need to continue operation when disconnected as mobile phones frequently have periods of signal loss (offline operation).

This application can be implemented completely on the edge, as explained previously, or partly on the cloud. For this application we have demonstrated the scalability of the Lasp prototype implementation up to 1024 nodes by using the *Amazon* cloud computing environment to simulate the edge network [26].

5 Future Directions

Building additional tools and support for a new generation of ultrascale edge applications is quite relevant and challenging. The varied nature of edge computing environments, which can combine small private clouds and data centers, specialized routing equipment and 5G towers, users desktops, laptops and even cellphones, to small things sensors and actuators, makes it a daunting task to build a single runtime support that can efficiently operate on all such devices and deal with their heterogeneity.

While we presented a set of tools and frameworks that can ease the development of ultrascale edge computing applications and services, these do not

cover all possible execution scenarios. That path to build such support requires not only the development of specialized runtimes for different edge settings, but also devising standard protocols and data representation models that allow the natural integration of different runtimes in a cohesive and effective edge architecture.

Current solutions for data replication and management are also unsuitable for the ultrascale that one is expected to find in emerging edge computing applications. The use of CRDTs to address the requirements of data management in this setting presents a viable approach. However, further efforts have to be dedicated in designing new and efficient synchronization mechanisms that can naturally adapt to the heterogeneity of the execution environment.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] J. Leitão, J. Pereira, and L. Rodrigues, "Epidemic broadcast trees," in *Symp. on Reliable Dist. Sys. (SRDS)*, pp. 301–310, Oct. 2007.

[3] J. Leitão, J. Pereira, and L. Rodrigues, "HyParView: A membership protocol for reliable gossip-based broadcast," in *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*, pp. 419–429, June 2007.

[4] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," Tech. Rep. RR-7687, INRIA, July 2011.

[5] P. S. Almeida, A. Shoker, and C. Baquero, "Delta state replicated data types," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 162–173, 2018.

[6] P. S. A. Carlos Baquero and A. Shoker, "Making operation-based crdts operation-based," in *Distributed Applications and Interoperable Systems - 14th IFIP WG 6.1 International Conference, DAIS 2014, Held as Part of the 9th International Federated Conference on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings*, pp. 126–140, 2014.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 2012.

[8] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, (New York, NY, USA), pp. 37–42, ACM, 2015.

[9] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*, pp. 29–36, ACM, 2012.

[10] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future generation computer systems*, vol. 29, no. 1, pp. 84–106, 2013.

[11] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[12] Cisco, "Cisco IOx Data Sheet," 2016.

[13] Dell, "Dell Edge Gateway 5000," 2016.

[14] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-peer computing," 2002.

[15] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.

[16] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[17] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, pp. 51–59, June 2002.

[18] C. Meiklejohn and P. Van Roy, "Lasp: A language for distributed, coordination-free programming," in *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming (PPDP 2015)*, pp. 184–195, Assoc. for Computing Machinery, July 2015.

[19] N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues, "Emergent structure in unstructured epidemic multicast," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, (Edinburgh, Scotland, UK), pp. 481 – 490, June 2007.

[20] V. Balegas, D. Serra, S. Duarte, C. Ferreira, M. Shapiro, R. Rodrigues, and N. Preguiça, "Extending eventually consistent cloud databases for enforcing numeric invariants," in *Symp. on Reliable Dist. Sys. (SRDS)*, (Montréal, Canada), pp. 31–36, IEEE Comp. Society, IEEE Comp. Society, Sept. 2015.

[21] P. E. O'Neil, "The escrow transactional method," *ACM Trans. Database Syst.*, vol. 11, pp. 405–430, Dec. 1986.

[22] A. Gotsman, H. Yang, C. Ferreira, M. Najafzadeh, and M. Shapiro, "'cause i'm strong enough: Reasoning about consistency choices in distributed systems," in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL 2016, (New York, NY, USA), pp. 371–384, ACM, 2016.

[23] D. D. Akkoorath, A. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. Preguiça, and M. Shapiro, "Cure: Strong semantics meets high availability and low latency," Tech. Rep. RR-8858, INRIA, Jan. 2016.

[24] A. van der Linde, P. Fouto, J. Leitão, N. Preguiça, S. Castiñeira, and A. Bieniusa, "Legion: Enriching internet services with peer-to-peer interactions," in *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, (Republic and Canton of Geneva, Switzerland), pp. 283–292, International World Wide Web Conferences Steering Committee, 2017.

[25] "Lasp: The missing part of Erlang distribution." `http://www.lasp-lang.org`. Accessed: 2018-04-27.

[26] C. Meiklejohn, V. Enes, J. Yoo, C. Baquero, P. Van Roy, and A. Bieniusa, "Practical evaluation of the Lasp programming model at large scale," in *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming (PPDP 2017)*, pp. 109–114, Assoc. for Computing Machinery, Oct. 2017.