LiquidAI: Towards an Isomorphic AI/ML System Architecture for the Cloud-Edge Continuum

Kari Systä[1], Cesare Pautasso[2], Antero Taivalsaari[1,3], Tommi Mikkonen[4]

[1] Tampere University, Tampere, Finland
kari.systa@tuni.fi
[2] USI, Lugano, Switzerland
cesare.pautasso@usi.ch
[3] Nokia Bell Labs, Tampere, Finland
antero.taivalsaari@nokia-bell-labs.com
[4] University of Jyväskylä, Jyväskylä, Finland
tommi.j.mikkonen@jyu.fi

Abstract. A typical Internet of Things (IoT) system consists of a large number of different subsystems and devices, including sensors and actuators, gateways that connect them to the Internet, cloud services, end-user applications and analytics. Today, these subsystems are implemented with a broad variety of programming technologies and tools, making it difficult to migrate functionality from one subsystem to another. In our earlier papers, we have predicted the rise of *isomorphic* IoT system architectures in which all the subsystems can be developed with a consistent set of technologies. In this paper we expand the same research theme to machine learning technologies, highlighting the need to use ML in a consistent and uniform fashion across the entire Cloud-Edge continuum.

Keywords: Isomorphic Software, Software Architecture, Internet of Things, IoT, Web of Things, WoT, Artificial Intelligence, AI, Machine Learning, ML, Software Deployment, Deployment in the Large, Programmable World

1 Introduction

A typical Internet of Things (IoT) system consist of a large number of computational elements. These elements include sensors and actuators, gateways that connect them to the Internet, cloud services, end-user applications running on mobile devices, and different kinds of analytics capabilities. Today, these computational elements are implemented with a broad variety of programming technologies and tools; for instance, IoT device development is still carried out primarily with traditionally embedded systems languages and tools, while web application and cloud backend development use an entirely different set of tools and technologies. This diversity makes it difficult to migrate functionality across the end-to-end system from one computational element to another. Instead, the functionality must be implemented using toolchains that are only applicable to certain types of components in the overall system. Any deployment changes typically imply a tedious re-implementation of the corresponding functionality.

Modern IoT systems and applications associated with them can generate and handle huge amounts of data. This has enabled Machine Learning (ML) and Artificial Intelligence (AI) in various use cases, ranging from smart home and smart city applications to healthcare, retail and industrial systems. Although IoT devices are generally assumed to be connected, not all the data from them can be moved to the cloud for processing because of privacy, latency or limited connectivity reasons. Thus, it is necessary to keep some computations close to the source of data, while other computations can run in the cloud. In many use cases there is a need to transfer data and computations seamlessly between different parts of the system, though. We have discussed this *cloud-edge continuum* in our earlier papers [22]. A recent literature study defined cloud continuum as *"an extension of the traditional cloud towards multiple entities (e.g., edge, fog, IoT) that provide analysis, processing, storage, and data generation capabilities"* [13]. Given the rapidly increasing use of ML technologies, we expect that the same technical challenges that apply to conventional computations shall emerge also in the context of ML technologies across the cloud-edge continuum.

The LiquidAI vision presented in this paper promises savings in the development effort by allowing flexible, dynamic, decentralized deployment of intelligent functions across the cloud-edge continuum. This is achieved by using a compatible set of technologies in all the subsystems, thus allowing different parts of the system to run the same code in an isomorphic fashion [20, 12]. The LiquidAI concept builds upon liquid software – a paradigm in which software applications can flow from one computing node to another in a seamless fashion [8, 23]. As a follow-up to our earlier work, we expand the liquid software concept to ML models that have an important role in today's IoT development and summarize the research challenges associated with it.

2 Background and Motivation

2.1 Artificial Intelligence in the Context of IoT

In recent years, processing and storage capabilities have grown dramatically, cloud computing has become commodity, data science has blossomed due to increasing amounts of data, and ML has emerged as everyday technologies even in devices with limited processing capabilities and resources such as mobile phones. These changes are leading us to a *Programmable World* [21], in which everyday things around us are becoming connected and programmable.

More broadly, the emergence of the IoT is acting as a catalyst for major changes in the development mindset. IoT developers must consider factors that are unfamiliar to many application developers today. Such factors include:

– multidevice programming of heterogeneous, diverse types of devices;
– the reactive, always-on nature of the overall system;

- intermittent, potentially unreliable nature of connectivity;
- the distributed, dynamic, and potentially migratory nature of software; and
- the general need to write software in a fault-tolerant and defensive manner.

In general, a typical IoT application is *continuous* and *reactive*. On the basis of observed sensor readings, computations get triggered (and retriggered) and may eventually result in various actionable events. In the context of the overall end-to-end system, programs are essentially *asynchronous, parallel* and *distributed*. In addition, the computational elements in the overall end-to-end system are typically heterogeneous and may possess very different processing capabilities and storage capacities.

In the context of this paper, we wish to highlight two areas especially:

- *Intelligence in the Edge*. In "classic" IoT systems, the majority of computation and analytics are performed in the cloud in a centralized fashion. However, in recent years there has been a noticeable trend in IoT system development to move intelligence closer to the edge (see, e.g., [10, 9]).
- *Rise of Swarm Intelligence*. In IoT systems that consist of a massive number of devices overall, device topologies can be expected to be highly dynamic and ephemeral (e.g. [18, 16]). This dynamism calls for technologies that can cope with dynamically changing *swarms* of devices and their dynamically evolving responsibilities at the holistic system level.

2.2 Liquid Software

Liquid software makes it possible for data and applications to move seamlessly between multiple devices and screens [23]. The concept of liquid software emerged originally in the context of multiple device ownership [8], referring especially to those use cases in which an individual user needs to use software applications in an uninterrupted fashion on different types of computing devices such as mobile phones, tablets, desktop computers and TVs – without having to explicitly install and/or launch applications anew or manually transfer data between those devices [23]. In recent years, liquid software technologies have expanded into IoT and other systems that do not necessarily have user interfaces [12]. In those contexts, liquid software typically refers to seamless transfer or migration of computations from one part of the system to another in order to best utilize the available computational resources. Moreover, liquid stream processing has been proposed, wherein data from Web-enabled sensors are gathered and sent for processing across a peer-to-peer cloud of computing peers [1].

The prerequisites of liquid software in the context of the IoT are (i) uniform API for accessing features of different subsystems, and (ii) a common runtime that is fast but small enough for embedded devices yet powerful enough to implement lightweight containers in order to deploy applications everywhere. In addition, (iii) an orchestrator is needed that will guide the deployment and potential migration of the different subsystems based on device proximity, connectivity and battery levels.

3 LiquidAI: Premises and Design Goals

3.1 The LiquidAI Vision

The requirement to run – and optionally also train – ML models and analytics algorithms in the edge increases the demand for consistent programming technologies in the overall end-to-end system. Our vision is that the required algorithms can be decomposed so that their components can be flexibly located and migrated in the cloud-edge continuum. Then, when requirements and network topology later evolve, the components can be relocated at different nodes in the IoT network. Moreover, security and privacy related issues must be reconsidered. Then, one can flexibly allocate and migrate functions across the cloud-edge continuum, considering available network bandwidth, latency, and computational resources.

To manifest an infrastructure capable of running such algorithms, the characteristics of liquid software need some revision. As stated above, the basic features include a uniform API, a common runtime, components that can be deployed and re-deployed at the cloud-edge continuum, and an orchestrator that can allocate and reallocate components. In addition, the creation of ML models requires reconsideration. Instead of creating individual ML models to process the given data, processing shall take place in a piecemeal fashion following the flow of data from the edge to the cloud, typically using techniques such as federated learning [11] and streaming data pipelines. To enable such piecemeal processing, monolithic ML models must be avoided and replaced with models that perform simple tasks and then forward the results to the next node for further processing.

Next, we present the design and research goals to reach the vision above.

3.2 Design Goals for LiquidAI

Uniform API. One of the key challenges in realizing the LiquidAI vision is the heterogeneous nature of the development languages, environments and tools, and the APIs and data formats that are associated with those technologies. There is a need for APIs to access resources on heterogeneous sets of devices, as well as for operations to manage data streams and various infrastructure features.

In the area of machine learning, application code needs to control the machine learning components (AI functions). API features shall be coherent and accessible from all relevant programming languages. Candidate base solutions in this area include the Web Thing API [3] and various service discovery protocols, which lack support for liquidity.

The research questions for a uniform liquid API include the following:

- What kind of an API allows liquid components to communicate with each other regardless of their current location?
- How can the API support access to streaming data in a unified fashion when the functions for processing the data streams may migrate as well?
- How wide a selection of programming languages can be supported and catered to with consistent APIs?

Common Runtime. By *runtime* we refer to technologies such as virtualization or virtual machines (VMs) that allow applications to run on top of the computing hardware – preferably independently of its physical architecture. In our context it is not enough to support built-in system applications only but also those that can be deployed and uninstalled dynamically.

The runtime shall support *strong* migration of software, where both the code and its current execution state can be transferred across different devices [4]. Such runtimes for liquid software have several and partially conflicting requirements: performance, hardware independence, support for various development paradigms, programming languages, and security. As concrete examples, WebAssembly and Node.js are both candidate runtimes for liquid applications. Due to virtualization, the runtime solutions that are used in the context of liquid software may not meet the performance requirements of machine learning applications. Moreover, runtimes used for ML applications may not support liquidity. Pathway to a compelling solution includes the following research challenges:

– What are the functional requirements of ML models towards the runtime?
– What are the performance and scalability requirements of ML models?
– How much can resource consumption be reduced if the runtimes for model training, validation, testing and inference in production are separated?

Decomposition and Deployment. Decomposition of software is essential for software maintainability, task allocation, and effective utilization of distributed computational elements. In addition to providing maintainability to software and its development, decomposition should split the software so that its components, optionally including ML features, can roam across the cloud-edge continuum. This places new requirements on decomposition. These observations have inspired a lot of research (e.g. [15, 5]). Today's state of the art approaches advocate the use of containers and orchestration to dynamically manage VM images and select specific node(s) according to algorithmic requirements (e.g. [2]). Unfortunately, this approach is rather heavyweight because of the use of containers, and thus lighter alternatives would be preferred (e.g. [17]).

Since ML models are key elements of LiquidAI systems, decomposing and deploying them across the cloud-edge continuum has received a lot of attention on a broad variety of use cases (e.g. [15]). However, isomorphic use of such functions is a new research direction that will be fundamental to achieving flexible composition of future IoT systems. Furthermore, when an application or component is deployed onto a target device, the software and required metadata have to be encapsulated in a proper way. The deployment, initialization and monitoring of the software may also require a specialized protocol, which should support both traditional and ML components. Based on the observations above, research questions related to decomposition include the following:

– What kind of decompositions are technically feasible for ML components, in relation to working practices in this field?
– Which solutions are compatible with the requirements of liquid deployment?

- What is the lifecycle of liquid functions, including both traditional and ML software?
- How to update liquid ML components with minimal disruption to system behavior?

Orchestration. Decomposition of LiquidAI systems into separate components introduces an orchestration challenge: how can multiple components ensure reliable end-to-end execution, provide scalability with large datasets and potentially massive amount of devices, as well as react to changing situations? Because IoT is largely about data, it is assumed that many of the target applications involve data streams. The foundation for orchestrating data streams across an IoT network has already been laid out in the Web Liquid Streams framework [1]. That framework helps developers create stream processing topologies and run them across a peer-to-peer network of connected devices and Web browsers. ML models are often also used for processing data via a series of pipes and filters (e.g. [19, 14]). In our vision, pipes are represented by connections in the stream processing topology, and filters are then dynamically deployed on the common runtime and communicate via the uniform API. We have identified the following research questions:

- How can the orchestrator control the migration of ML models?
- How to manage and monitor ML models in liquid context?
- How to ensure reliability and trustworthiness a distributed system where stateful components may roam between locations?
- How can performance, memory and bandwidth be optimized if ML models are partitioned across a pipeline operating on a common data stream?
- What is state for processing streaming data with AI functions?

Liquid Features. In our earlier work [6] we have investigated the architecture and design issues of liquid software. Much of the earlier research has focused on user interfaces and especially user interface adaption to a *set of* devices with different usage modalities (phones, tablets, laptops, TVs, etc.) In LiquidAI, most system components are headless (have no user interface), and therefore many aspects need to be revisited and extended. First, the handling of state – seamless data transfer and state synchronization is a key characteristic of liquid software. In LiquidAI, the viewpoint might be different given that the presence of data streams and learning – especially incremental learning – can be regarded as a form of state. Second, while the LiquidAI concept has no user sessions roaming across devices, applications still need to take full advantage of all devices that they run across. There is a need to optimize performance, including both computing and network traffic, with tradeoff factors such as power consumption and latency. The research questions related liquid functionality include:

- Do machine-learning components have a state?
- How can liquid components adapt to different hardware characteristics?

Security and Privacy. Because many of the LiquidAI systems will deal with sensitive data, they should be decomposed in such a fashion that privacy and access rules are enforced. So far, relatively small amount of research has been directed into security features needed by liquid software at large given that the prime use case has been personal computing experience in which all the devices have the same owner. Furthermore, security related concerns in IoT are common in general (e.g. [24, 7]), which underlines research needs on this topic. Important research questions include the following:

- How to provide security for liquid software across the cloud-edge continuum?
- How to ensure privacy guarantees with a set of ML components learning from or processing data streaming pipelines?
- How containers for liquid software can prevent leaks of sensitive data?

4 Conclusions

In this paper, we presented a vision and a tentative research agenda for LiquidAI – a framework in which machine learning and data streaming can coexist and be flexibly orchestrated in IoT networks. The vision extends our earlier work on liquid software applications, refocusing the technology to end-to-end IoT systems. As part of this work, we have formulated the concept and identified a set of research questions that must be answered in order to realize the vision. We hope that this paper, for its part, encourages the Web engineering research community to collaborate with us on finding answers to these research questions.

Acknowledgments.This work has been supported by Business Finland (project LiquidAI, 8542/31/2022).

References

1. Babazadeh, M., Gallidabino, A., Pautasso, C.: Decentralized stream processing over web-enabled devices. In: European Conference on Service-Oriented and Cloud Computing. pp. 3–18. Springer (2015)
2. Debauche, O., Mahmoudi, S., Mahmoudi, S.A., Manneback, P., Lebeau, F.: A new edge architecture for AI-IoT services deployment. Procedia Computer Science **175**, 10–19 (2020)
3. Francis, B.: Web Thing API. https://webthings.io/api/, retrieved 2023-01-25
4. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding code mobility. IEEE Transactions on software engineering **24**(5), 342–361 (1998)
5. Gallidabino, A., Pautasso, C.: The LiquidWebWorker API for horizontal offloading of stateless computations. Journal of Web Engineering **17**, 405–448 (March 2019). https://doi.org/10.13052/jwe1540-9589.17672
6. Gallidabino, A., Pautasso, C., Mikkonen, T., Systä, K., Voutilainen, J.P., Taivalsaari, A.: Architecting liquid software. Journal of Web Engineering **16**(5-6), 433–470 (September 2017). https://doi.org/10.26421/JWE16.5-6, http://www.rintonpress.com/journals/jweonline.html#v16n56

7. Gurunath, R., Agarwal, M., Nandi, A., Samanta, D.: An overview: security issue in IoT network. In: 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud). pp. 104–107. IEEE (2018)
8. Hartman, J.J., Bigot, P.A., Bridges, P., Montz, B., Piltz, R., Spatscheck, O., Proebsting, T.A., Peterson, L.L., Bavier, A.: Joust: A platform for liquid software. Computer **32**(4), 50–56 (1999)
9. Keshavarzi, A., van den Hoek, W.: Edge intelligence—on the challenging road to a trillion smart connected IoT devices. IEEE Design & Test **36**(2), 41–64 (2019)
10. Liu, Y., Peng, M., Shou, G., Chen, Y., Chen, S.: Toward edge intelligence: Multiaccess edge computing for 5G and internet of things. IEEE Internet of Things Journal **7**(8), 6722–6747 (2020)
11. Ludwig, H., Baracaldo, N.: Federated Learning: A Comprehensive Overview of Methods and Applications. Springer (2022)
12. Mikkonen, T., Pautasso, C., Taivalsaari, A.: Isomorphic Internet of Things architectures with web technologies. Computer **54**(7), 69–78 (2021)
13. Moreschini, S., Pecorelli, F., Li, X., Naz, S., Hästbacka, D., Taibi, D.: Cloud continuum: The definition. IEEE Access **10**, 131876–131886 (2022)
14. Pääkkönen, P., Pakkala, D.: Reference architecture and classification of technologies, products and services for big data systems. Big data research **2**(4), 166–186 (2015)
15. Peltonen, E., Ahmad, I., Aral, A., Capobianco, M., Ding, A.Y., Gil-Castineira, F., Gilman, E., Harjula, E., Jurmu, M., Karvonen, T., et al.: The many faces of edge intelligence. IEEE Access **10**, 104769–104782 (2022)
16. Puschmann, D., Barnaghi, P., Tafazolli, R.: Adaptive clustering for dynamic IoT data streams. IEEE Internet of Things Journal **4**(1), 64–74 (2016)
17. Raghavendra, M.S., Chawla, P.: A review on container-based lightweight virtualization for fog computing. In: 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO). pp. 378–384. IEEE (2018)
18. Seeger, J., Deshmukh, R.A., Sarafov, V., Bröring, A.: Dynamic IoT choreographies. IEEE Pervasive Computing **18**(1), 19–27 (2019)
19. Sena, B., Garcés, L., Allian, A.P., Nakagawa, E.Y.: Investigating the applicability of architectural patterns in big data systems. In: Proceedings of the 25th Conference on Pattern Languages of Programs. pp. 1–15 (2018)
20. Strimpel, J., Najim, M.: Building Isomorphic JavaScript Apps: From Concept to Implementation to Real-World Solutions. O'Reilly Media (2016)
21. Taivalsaari, A., Mikkonen, T.: A roadmap to the programmable world: software challenges in the IoT era. IEEE software **34**(1), 72–80 (2017)
22. Taivalsaari, A., Mikkonen, T., Pautasso, C.: Towards seamless IoT device-edge-cloud continuum. In: International Conference on Web Engineering. pp. 82–98. Springer (2021)
23. Taivalsaari, A., Mikkonen, T., Systä, K.: Liquid software manifesto: The era of multiple device ownership and its implications for software architecture. In: 2014 IEEE 38th Annual Computer Software and Applications Conference. pp. 338–343. IEEE (2014)
24. Zhang, Z.K., Cho, M.C.Y., Wang, C.W., Hsu, C.W., Chen, C.K., Shieh, S.: IoT security: ongoing challenges and research opportunities. In: 2014 IEEE 7th international conference on service-oriented computing and applications. pp. 230–234. IEEE (2014)