This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MIC.2021.3129271, IEEE Internet Computing

Edge-based autonomous management of vertical farms

Adrian Jandl, Pantelis A. Frangoudis, and Schahram Dustdar
Distributed Systems Group, TU Wien, Vienna, Austria

***Abstract*—Vertical farming is the practice of growing crops vertically to increase total yield for a given space and is one approach towards sustainable food production. Concerns related with its operational costs, and the need to optimize plant growth parameters in a controlled environment call for advanced use of IoT technologies to develop low-cost mechanisms for continuous monitoring and optimization of vertical farming processes. This article lays the groundwork for such mechanisms, providing an extensible, edge-centric architecture for IoT-supported autonomous vertical farm monitoring and management. We study alternative deployment strategies for it, exploring the design and performance implications of using LoRaWAN as the device connectivity substrate. We show experimentally that it is possible to handle vertical farm monitoring workloads corresponding to thousands of IoT devices, even when operating purely on minimal edge compute infrastructure, making it feasible to support the management of vertical farms cheaply and at scale.**

BY THE YEAR 2050 the worldwide population is projected to be above 9 billion [1]. A key challenge for this century will be to provide all people on earth with enough food. This will prove especially difficult, as climate change may reduce the amount of farmable land. One proposed solution is vertical farming [2], which is recently attracting interest [3]. Its basic principle is growing as much food as possible on as small a space as possible, by growing crops above each other rather than next to each other. This can be done both outdoors by using natural light, or indoors by utilizing artificial lighting, a significant cost factor.

The opportunities of vertical farming to make better use of space and better control factors that affect production come directly with some new challenges. From a physical perspective, growing plants vertically hinders their manual inspection and reachability by farmers. Space limitations, besides cost, also affect how sensing and communication infrastructure is deployed; for example, there is a need to minimize cabling, thus wireless sensor connectivity is preferred. Vertical farming is often associated with planned and connected indoor installations, but under specific circumstances such as in underground or remote outdoor farms, Internet connectivity cannot always be assumed. Therefore, farm monitoring and control logic needs to be executed in place, at the edge, instead of in the cloud.

Studies show [4] that energy consumption (for indoor deployments) and labor cost are the highest contributing operational expenditure factors. At the same time, a multitude of heterogeneous monitoring data directly originating from IoT devices are generated in a vertical farm, and there are, respectively, plenty of parameters that a farmer can directly influence and which determine plant growth and resource consumption. This calls for low-cost and high-volume monitoring data management, which is critical for optimized control of the vertical farming process, and in order to produce new domain knowledge about optimal

farming practices and procedures potentially not yet well understood.

Towards addressing such challenges, we presenting a modular, service-based architecture and end-to-end system for the management of vertical farm installations supported by IoT technologies. Our design is derived with extensibility in mind and is connectivity technology agnostic. However, physical deployment and networking aspects are critical. We therefore evaluate different networking scenarios to support our scheme, with a focus on Low-Power Wide Area Networks (LPWAN). We build a prototype on commodity edge computing devices, featuring state-of-the-art stream processing technology and integrating IoT devices connected, among others, via LoRaWAN. Our experiments reveal that, from a computation and communication perspective, low-end edge computing devices are suitable candidates to implement autonomic vertical farm management at a low cost and at scale.

Architecture for vertical farm management

Key requirements for a system to support the operation of a vertical farm include the ability (i) to seamlessly integrate a multitude of heterogeneous sensors, typical of the different parameters and phenomena to be monitored, and, in turn, (ii) provide custom analysis and decision making logic for different aspects of the farming process, operating on monitoring data streams and potentially integrating domain knowledge. Furthermore, (iii) it is desirable that the system can function on resource-constrained compute infrastructure, such as Single Board Computers (SBCs) like Raspberry Pi (RPi), which we expect to be typical of edge-assisted vertical farming installations given their low cost, reduced power requirements, and small form factor; these are important when space and power consumption matter.

Driven by these requirements, we present the components of our architecture (Figure 1). Our design aligns with the philosophy of autonomic computing [5], thus our functional blocks map to the elements of a Monitor-Analyse-Plan-Execute (MAPE) loop.

Sensing Module

One or more Sensing Modules (SMs) are in charge of feeding sensor measurements into our system. A SM may also include actuation components. We have applied a *sensor gateway* approach; the SM aggregates and ingests data to a publish/subscribe system which delivers them to a *Controller* and various *Decision Modules* (DMs). This approach allows to collect data from sensors which are controlled via different systems and programming languages, and over different connectivity forms. It also provides a single entry point into our stream processing pipeline and helps with extensibility. The SM acts as a wrapper for the sensor gateway and uses a unified interface to communicate with the Controller. The internals of the gateway implementation are thus abstracted. Furthermore, the SM implements configuration and alert functions, which (i) expose the respective interface to receive notifications or changes in sensor settings, and (ii) act upon them, dealing with the particular sensor/actuator communication interface. For example, sensors connected over LoRaWAN do not have an IP endpoint and the SM should interface with the LoRaWAN network to push a downlink message.

Controller

The Controller is in charge of routing data across the various system entities and performing format translations, if necessary. Configuration changes mainly originate at the User Interface (UI) and are pushed to the SMs via the Controller, whose configuration module provides an interface where SMs can register to receive alarms and configuration changes. Alarms and other notifications originate from DMs. When raised, the Controller needs to push them to the UI and the SMs.

Decision Module

A DM is responsible for data analysis and action planning. It subscribes to, consumes, and analyzes data produced by SMs. If an alarm should be raised or a notification be generated, it publishes the respective message. The system should support common threshold-based alarms and generic notifications (e.g., indications that the current maximum value of a sensor has changed) out-of-the-box, while multiple DMs, potentially providing sophisticated analytics and decision

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MIC.2021.3129271, IEEE Internet Computing

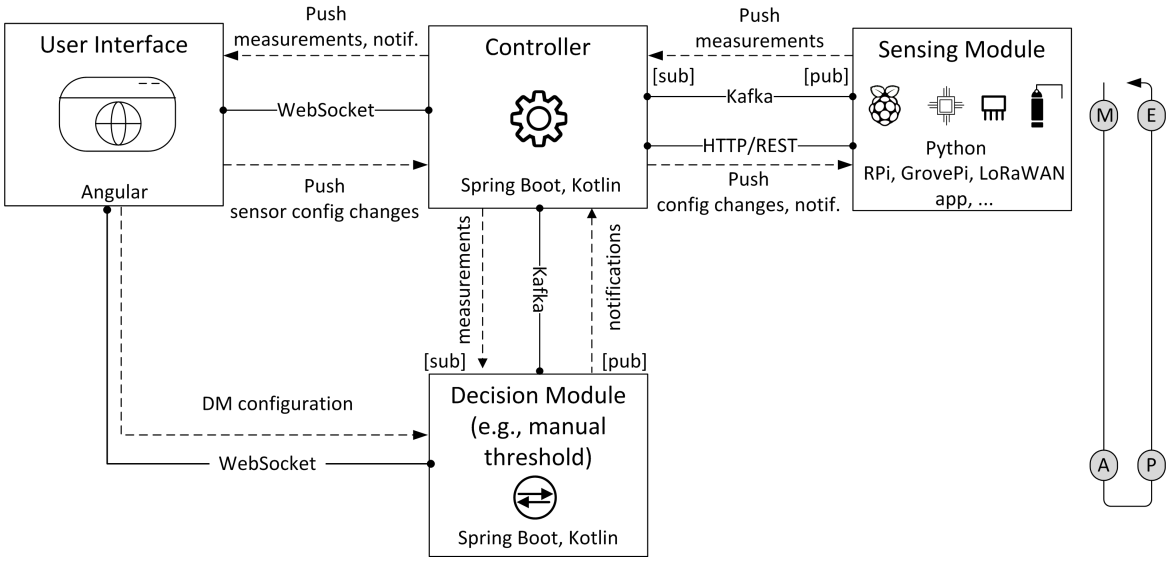

Figure 1: Architecture components and interactions in the context of a Monitor-Analyze-Plan-Execute loop [5]. The technologies used in our implementation are also depicted.

making, can coexist as *pluggable* modules. Moreover, each DM can have interfaces of its own, e.g., for the farmer to manually provide domain knowledge (for example, threshold values). These are left to the architects of the various DMs.

Network topologies, technologies, and deployment strategies

The presented software architecture is designed to be agnostic to the underlying topology and connectivity technologies. However, these have significant implications when it comes to implementing, deploying and operating such a system. We analyze different deployment scenarios and network topologies where our system could be run; combinations of these scenarios are also possible.

A. On-site hosted (Wireless) Local Area Network

In a small-scale farm that has good power supply possibilities, the system can utilize Wi-Fi internally for the communication between the SM, the Controller, the DMs and the UI, all of which can be hosted at edge computing nodes on-site and within the same network. The SM-internal communication with the sensors can be accomplished via other means, such as over Bluetooth, Wi-Fi, a wired connection or a sensor-specific hardware interface. Given the range of Wi-Fi, multiple wireless access points may be required to cover the farm. Importantly, this setup comes with no external hosting costs and relies on no third parties, enabling autonomous disconnected operation. The farmer only has on-site access to the system in this case.

To allow remote farm monitoring, a public IP endpoint at the farm is typically necessary. Then, which components of the architecture are deployed at the edge is a matter of the desired level of cloud reliance. For example, it is possible to deploy the SMs, Controller, and DMs on-site and keep the UI in the cloud, or follow a more cloud-centric approach, splitting the SM to a local (edge) gateway component that collects sensor readings and a remote (cloud) one which receives sensor data from the gateway and publishes them.

B. Use of LPWAN technology

A different approach to device connectivity is via LPWAN. This allows to host the monitoring system at an aggregation point at the edge or in the cloud, reducing network infrastructure requirements in the farm. Our LPWAN technology of choice is LoRaWAN, due to its potential to operate a full end-to-end private network without relying on a network provider, and its ability to integrate well with both edge and cloud computing resources [6]. Here, sensors (or sensor gateways where multiple sensors are attached) are equipped with LoRaWAN radio interfaces to emit their

readings.

According to the LoRaWAN specification [7], end devices broadcast data over the LoRa physical layer, which are received by LoRaWAN gateways. A LoRaWAN gateway then forwards the data to a Network Server (NS), which is responsible for handling the state of the network, including device join requests, MAC-layer operations such as packet deduplication and downlink scheduling, and pushing uplink data to an Application Server (AS) component. The AS implements the application layer of the LoRaWAN stack. It provides various integration mechanisms with IoT application services, and also handles application-level security, such as end-to-end data encryption.

LoRaWAN operates in unlicensed spectrum, but, depending on the region, it is subject to channel dwell time restrictions (as in the US) or *duty cycling* regulation (which is the case in the EU). For example, an IoT device may not be allowed to transmit for more than 0.4 s over a 20 s period at a given frequency, or it may have to apply a 99% sleep time and 1% transmission time. This limits how many measurements a device can send in a given timeframe. Also, uplink latency for LoRaWAN is higher than that of Wi-Fi, but vertical farming typically does not come with real-time constraints. Notably, in vertical farming it is sometimes the case that cameras are used as sensors. In this case, the limited uplink bandwidth of LoRaWAN mandates that data (images) are preprocessed on-device so that only extracted features are transmitted, to be used as input to DMs. Even so, the very limited frame payload size may necessitate that fragmentation is handled at the application level, making it cumbersome or infeasible to use LoRaWAN.

Downlink communication also faces limitations. One example is the change of SM configuration values, such as the measurement interval. There are three LoRaWAN device classes, with Class A being the most widespread, inexpensive, and with the minimum energy requirements. A Class A device can only receive a downlink message during one of the two short *receive windows* (RX1 and RX2) that it opens a specified time after an uplink [7]; by default, RX1 and RX2 open 1 s and 2 s, respectively, after an uplink has been completed. This means that the latency for a notification or configuration change to apply is determined by the uplink interval.

Table 1: Rating of deployment scenarios on a five-point scale.[a]

Scenario	Ease of network setup	Infrastructure independence	Remote monitoring	Compute power
Private LAN	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆	⋆⋆
Public LAN	⋆⋆⋆⋆	⋆⋆	⋆⋆⋆⋆	⋆⋆⋆(⋆⋆)
LoRaWAN	⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆(⋆⋆⋆)

[a] Under the compute power category, stars in parentheses represent a potential increase in the ranking of a scenario when it is permissible to deploy architecture components in the cloud, i.e., when disconnected operation is not mandated.

On the positive side, LoRaWAN poses less infrastructure requirements, combined with extended battery life of IoT devices. With a cloud-based deployment of the architecture components, or by deploying a connected LoRaWAN gateway and the rest of the architecture at a single edge location in the farm, remote monitoring for the farmer is facilitated. Importantly, a LoRaWAN gateway has a coverage radius in the order of kilometers. However, an increased setup effort typically by experienced IT personnel might be needed. A LoRaWAN-based topology for our architecture is shown in Figure 2.

Comparison

While choosing the appropriate deployment strategy is a multi-faceted decision, this being a matter of preference and the particularities of the farm settings, we attempt to provide a structured comparison which, combined with the priorities of the system operator, can guide the latter's choices. We rate the various scenarios along four dimensions on a five-point scale (Table 1).

Using a private WLAN ranks better regarding the ease of network setup. Providing external access might require some extra effort. LPWAN scenarios may require the additional configuration of a gateway and the respective network stack. Despite that, LPWAN-based deployment ranks better in terms of infrastructure independence. (This applies when an end-to-end LoRaWAN private network is deployed. Other options are possible [6] but not discussed here.) This owes to the fact that IoT devices can operate on battery for prolonged periods and that a single gateway can cover a wide area. For the same reason,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MIC.2021.3129271, IEEE Internet Computing

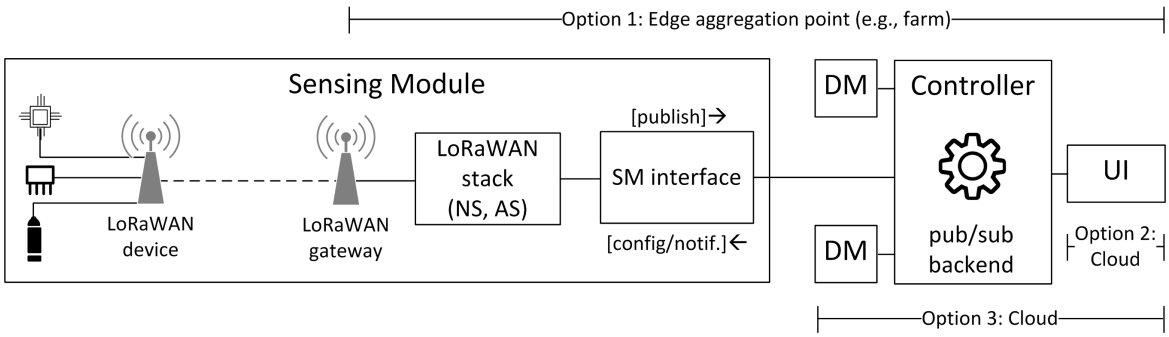

Figure 2: Deployment featuring LoRaWAN. Different hosting scenarios are shown, with different levels of cloud dependence each; from hosting everything at an edge aggregation point (option 1), to pushing the Controller, DMs, and UI to the cloud (option 3). Other options are also possible.

LoRaWAN-based topologies score better in terms of remote monitoring. Even if the monitoring system is not Internet-accessible, the farmer may have access to it from a single edge aggregation point (e.g., where the LoRaWAN gateway is co-located with the rest of the system components) or the cloud. Also, a LoRaWAN-based setup has benefits for multi-site farm installations where different sites are in the same geographical area and sensors from multiple physical locations feed monitoring data into a single remote controller in a star-like topology.

Computation-wise, hosting everything at the edge (i.e., on-premise) is naturally more limited. The lower ranking reflects the need to appropriately dimension the compute resources necessary, since these are provided by dedicated edge hardware such as SBCs. By mandating disconnected operation, this also constrains the cloud-based scaling capabilities of such setups. In contrast, other scenarios may rank better in this respect, depending on the extent to which they are using cloud resources to host application components.

Implementation

We have implemented our architecture relying on Apache Kafka for our publish/subscribe backend, which provides a scalable stream processing engine at the core of our system. Once a measurement is taken, it is published by the SM (written in Python) to a specific Kafka topic (`vf-sensor-topic`) using the `kafka-python` library. The SMs register with their IP address to an HTTP endpoint of the Controller to be notified of alerts and configuration changes. The SM-sensor interface is internal and depends on the particular sensors used. To demonstrate the flexibility of our design, we implemented two different types of sensor communication: (i) a solution where the sensor is bundled on the same hardware as the SM; (ii) a solution based on LoRaWAN. In both cases, the same unified interface is used to communicate with the Controller.

Our LoRaWAN-based proof of concept uses end-devices with temperature and other sensors. A reading is transmitted over LoRaWAN to a gateway hosted by a RPi. The full LoRaWAN network server stack, for which we used the ChirpStack (https://www.chirpstack.io/) open-source implementation, is co-located with the gateway device. Uplink frames traverse the NS and the AS and, via an integration layer, are eventually pushed to a south-bound (internal) REST API endpoint of the SM. On the downlink, the SM enqueues the payload (e.g., sensor configuration) to an API endpoint of the AS, which forwards it to the NS for delivery.

The Controller is implemented in Kotlin using the Spring Boot framework, and consumes data published to `vf-sensor-topic` using the Kafka Consumer Java API. These are forwarded to our web-based UI over a WebSocket. If an alert is published, the Controller propagates it to the UI and notifies via REST the SM that an alarm is to be raised.

We developed two DMs which consume and operate on sensor data streams and publish notifications. The *min-max* DM publishes an alarm whenever a new maximum/minimum value is reached

for a sensor. The *manual threshold* DM allows users with domain knowledge to set manually critical threshold values for sensors via the UI.

Evaluation

We carry out an experimental campaign, striving to draw realistic figures on the capacity of our system to handle vertical farm monitoring workloads, especially when operating atop low-end edge computing hardware. This is directly linked with the scale of farms we can manage, and the associated infrastructure costs. We perform experiments where two types of compute devices are involved: a RPi 2 Model B (900MHz quad-core ARM Cortex-A7 CPU; 1 GB RAM) and a host with an AMD Ryzen 5 3600X processor (6 CPU cores at 3.80GHz; 32 GB RAM).

Workload processing capacity

We first focus on the rate at which an SM (producer) running on a RPi can inject sensor readings to the system. This setup is characteristic of cases where the SM runs on a single SBC and acts as the sensor gateway which publishes sensor readings to the Controller, aggregating a large number of devices. The rest of the software stack runs on a separate powerful host. To avoid hitting network bottlenecks, devices were connected via 1 GB Ethernet. We found that the Kafka producer on a RPi can emit more than 1000 measurements/s with 100% delivery rate. This implies that many important parameters in the operation of a vertical farm, such as light intensity, pH levels, and others, can be monitored at a high frequency. This adds to the feasibility of our design and implementation.

Vertical farm monitoring over a private LoRaWAN network

We then turn our attention to the use of LoRaWAN for IoT device connectivity. We are particularly interested in the deployment of a full private LoRaWAN network to support a vertical farm, and since low cost is a prime concern, we aim to do so with minimal equipment. Thus, we experiment with an all-in-one setup, where the components of the LoRaWAN stack are co-located with the LoRaWAN gateway and are executed on top of the same edge device, which also hosts the SM and the Controller. Our testbed setup includes a Libelium Smart Water end device, with a Microchip RN2483 LoRa radio module on an ATmega1281 MCU. Regarding the gateway, we attached a Dragino PG1301 LoRaWAN concentrator with a Semtech SX1301 baseband unit and SX1257 RF front-end on the RPi which also hosts the ChirpStack LoRaWAN stack. In the experiments that follow, each transmission carries a 4-byte payload. The spreading factor was set to 7 and the uplink channel bandwidth was 125 KHz, operating at the EU 863-870 MHz frequency band.

Latency We first measure the round trip time of a sensor reading from the IoT device to the ChirpStack network stack and back. This communication is internal to the SM: when the AS receives a reading, it pushes it to the internal HTTP endpoint of the SM, which then publishes it to the Controller.

We measure the time to (i) sense a temperature value, (ii) transmit a frame over LoRaWAN, and (iii) receive a response (acknowledgement and/or queued downlink frame). Transmitting also involves powering on the LoRaWAN radio and configuring it with the authentication state (i.e., the following session parameters: device address, network session key, application session key) established when the device joins the network using over-the-air-activation (OTAA). In this process, which takes $\sim$7.84 s on average (Figure 3), the call to transmit the frame and receive the downlink data accounts for 2.5 s. It is interesting to see that the 2 s receive window that opens after a LoRaWAN transmission is manifested in this value.

Figure 3 (bottom) shows that the majority of the time (48%) is spent to power on the LoRaWAN radio module and re-configure it with the parameters set up when the device joined the network, followed by the actual transmission of data at 33%. Powering on/off the radio module before/after each transmission and having to re-configure session parameters each time is optional, but is applied to save on energy consumption. There is a non-negligible amount of time spent in `sleep()` calls after each command sent from the host MCU to the radio module. This takes place over a UART hardware interface and the sleep guards are in place to ensure that commands execute properly. This is specific to the hardware and software of our testbed.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MIC.2021.3129271, IEEE Internet Computing

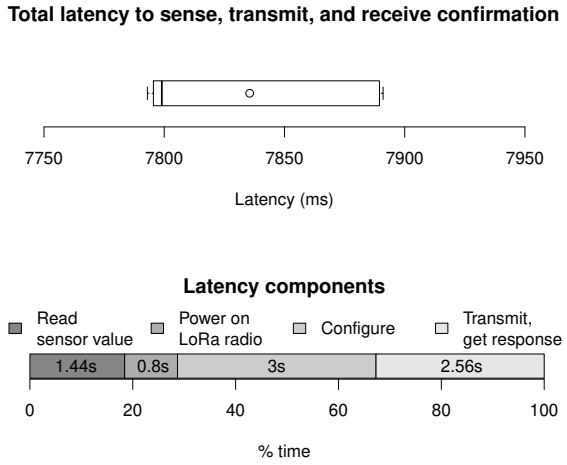

Figure 3: Top: Total latency to communicate a sensor reading over LoRaWAN, from sensing to receiving a confirmation. The median and mean are shown with a line and circle point, respectively. Minimum, maximum, first and third quartile values are shown. Bottom: Breakdown of the total latency. After powering on the LoRaWAN module, configuration refers to setting session parameters which were derived when the device joined the network.

Message processing throughput We used the ChirpStack Simulator to inject load in the LoRaWAN network stack in the form of simulated uplink frames, and measured how much it takes for a `sendConfirmed` call from a real device to complete. Confirmed transmission is an optional feature, where the NS acknowledges reception. If an acknowledgement is not received after a number of retries (seven, in our tests), the frame is considered lost. Our tests showed that the LoRaWAN stack, when deployed on a RPi, has a maximum uplink message processing capacity of less than 60 messages/s. When the injected load reached 60 messages/s, we observed an abrupt latency increase from 2.5 s to approximately 13 s. This is because the increased workload on the LoRaWAN network stack leads to increased time to process uplink frames. These latencies, in turn, often cause the NS to fail to deliver an acknowledgement for an uplink frame in time, i.e., within the two receive windows that follow, thus leading to retransmissions, which add to the latency experienced by the device. Additionally, $\sim$25% of the messages were lost, which further indicates that we reached capacity.

Implications of our results Using LoRaWAN as the device connectivity technology for vertical farm monitoring is promising. Latencies in the order of few seconds are still acceptable since no hard real time requirements typically apply. Regarding downlink traffic, using Class A LoRaWAN devices, latency depends on the uplink message transmission interval. If there is a message queued for delivery to a sensor device (such as a configuration change), this will be delivered as a response to the next uplink message.

We derived a limit of 60 uplink messages/s when the full LoRaWAN stack is hosted at an edge device. While this might look modest, it corresponds to thousands of sensors generating readings at realistic frequencies of once per few minutes each. Importantly, this value is associated with the processing capabilities of our reference edge device. Using a powerful host or VM executed at edge or centralized clouds would yield significantly better capacity. Furthermore, the LoRaWAN stack can be scaled horizontally, provided that more edge compute resources are available to balance the load. Our results can help the system designer to directly derive the amount of such resources necessary (e.g., number of SBCs at a local edge cluster) to support a target workload.

Cost considerations

We built our proof of concept at a cost of (prices as of November 2020) 40€ for the SBC and 130€ for the sensor kit used, which included a number of temperature, humidity and light sensors. For a LoRaWAN-based installation, the gateway radio hardware accounted for 130€, while an end-device LoRaWAN radio is at the order of 10€ (this can be combined with an MCU platform or an SBC controlling multiple sensors). Since it is feasible computationally to monitor a large-scale vertical farm installation with a single SBC, eventually it may be IoT device procurement that will dominate setup costs, rather than compute infrastructure. Subscription costs may also surface, for instance to provide Internet access to the farm, or if another device connectivity approach is selected, such as using NB-IoT or a commercial

LoRaWAN network operator. For a detailed cost-driven comparison of different LPWAN-based connectivity options and deployment strategies, the reader is referred to our prior work [6].

Related work

Vertical farming can be viewed as a case for precision agriculture [8]. To support precision agriculture, appropriate IoT service architectures need to be in place [9], [10], [11] but currently the majority are cloud-centric. In the context of vertical farming, these should handle data from a multitude of sensing technologies. Cameras as sensors, for example, are commonplace [12], [13], while the use of sensing-capable drones, as already applied in traditional outdoor settings [14] is promising. Our design facilitates the integration of such technologies.

Prototypes of IoT-supported farming are emerging. SmartFarmNet [15] relies on semantic web technologies and aims to address device heterogeneity. FarmBeats [16] introduces an IoT base station connected over TV white spaces and uses drones combined with ground sensors for farm mapping.

IoT data processing is traditionally cloud-based, but data volume and velocity stress the communication infrastructure and necessitate significant resources for centralized processing. This motivates pushing computation closer to data sources, i.e., the IoT device domain, giving rise to edge computing [17]. Edge computing use cases permeate the whole IoT space, and, recently, smart agriculture. This is manifested in cloud-focused systems for vertical/soil-less farming [18], [19] with similar motivation as ours (albeit different technical approach and research focus), which can exploit edge/fog resources for certain tasks. O'Grady et al. [20] survey various edge computing approaches in agriculture. They conclude that most of them have a level of cloud dependence, identify the lack of Internet connectivity as a key limitation, and advocate for a service model based on delay tolerance. Our work is in line with this spirit. We support various levels of cloud dependence, facilitating complex stream processing and analytics at the edge. Most importantly, our service-oriented design natively supports publish-subscribe asynchronous communication between components, a prerequisite for delay- and disruption-tolerance.

Conclusion

Vertical farming has emerged as an approach to sustainable food production, but faces challenges related with operational costs, monitoring, and management. We have shown how such challenges can be addressed by combining Service-Oriented Architecture, advances in IoT connectivity, and low-cost edge computing technology. This is a promising result in an effort to reduce the price of entry to vertical farmers. Our analysis of different deployment strategies and experimental results can further serve as a basis for dimensioning the compute infrastructure necessary for IoT-driven vertical farm management from the edge.

REFERENCES

1. United Nations, Department of Economic and Social Affairs, Population Division, *World Population Prospects 2019: Highlights*. New York: United Nations, 2019.
2. D. Despommier, "The vertical farm: controlled environment agriculture carried out in tall buildings would create greater food safety and security for large urban populations," *Journal für Verbraucherschutz und Lebensmittelsicherheit*, vol. 6, no. 2, pp. 233–236, 2011.
3. M. Butturini and L. F. Marcelis, "Chapter 4 - vertical farming in europe: Present status and outlook," in *Plant Factory*, 2nd ed., T. Kozai *et al.*, Eds. Academic Press, 2020, pp. 77–91.
4. C. Zeidler *et al.*, "Vertical farm 2.0: Designing an economically feasible vertical farm - a combined european endeavor for sustainable urban agriculture," Association for Vertical Farming, Tech. Rep., 2017, white Paper. [Online]. Available: https://elib.dlr.de/116034/
5. J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
6. P. A. Frangoudis *et al.*, "Connectivity technology selection and deployment strategies for IoT service provision over LPWAN," *IEEE Internet Comput.*, vol. 25, no. 1, pp. 61–70, 2021.
7. LoRa Alliance, *LoRaWAN 1.1 Specification*, Oct. 2017.
8. N. Zhang *et al.*, "Precision agriculture—a worldwide overview," *Computers and Electronics in Agriculture*, vol. 36, no. 2, pp. 113–132, 2002.
9. M. S. Farooq *et al.*, "A Survey on the Role of IoT in Agriculture for the Implementation of Smart Farming," *IEEE Access*, vol. 7, pp. 156 237–156 271, 2019.
10. T. Ojha *et al.*, "Internet of Things for Agricultural Applications: The State-of-the-art," *IEEE Internet of Things Journal*, 2021, in press.

11. J. López-Riquelme *et al.*, "A software architecture based on FIWARE cloud for precision agriculture," *Agricultural Water Management*, vol. 183, pp. 123–135, 2017.
12. M. L. Tenzer and N. C. Clifford, "A digital green thumb: Neural networks to monitor hydroponic plant growth," in *Proc. SIEDS*, 2020.
13. L. Zhang *et al.*, "Growth monitoring of greenhouse lettuce based on a convolutional neural network," *Horticulture Research*, vol. 7, no. 124, 2020.
14. P. Tripicchio *et al.*, "Towards smart farming and sustainable agriculture with drones," in *Proc. IE'15*, 2015.
15. P. P. Jayaraman *et al.*, "Internet of things platform for smart farming: Experiences and lessons learnt," *Sensors*, vol. 16, no. 11, 2016.
16. D. Vasisht *et al.*, "Farmbeats: An IoT platform for data-driven agriculture," in *Proc. USENIX NSDI*, 2017.
17. M. Gusev and S. Dustdar, "Going back to the roots—the evolution of edge computing, an IoT perspective," *IEEE Internet Computing*, vol. 22, no. 2, pp. 5–15, 2018.
18. I. Haris *et al.*, "CPS/IoT Ecosystem: Indoor Vertical Farming System," in *Proc. IEEE ISCT*, 2019.
19. M. A. Zamora-Izquierdo *et al.*, "Smart farming IoT platform based on edge and cloud computing," *Biosystems Engineering*, vol. 177, pp. 4–17, 2019.
20. M. O'Grady *et al.*, "Edge computing: A tractable model for smart agriculture?" *Artificial Intelligence in Agriculture*, vol. 3, pp. 42–51, 2019.

Adrian Jandl received his Master's Degree in Software Engineering and Internet Computing from TU Wien, Vienna, Austria. Contact him at adrian.jandl@gmail.com.

Pantelis A. Frangoudis is a post-doctoral researcher with the Distributed Systems Group, TU Wien, Vienna, Austria. He is the corresponding author of this article. Contact him at pantelis.frangoudis@dsg.tuwien.ac.at.

Schahram Dustdar is a Full Professor of Computer Science (Informatics) with a focus on Internet Technologies heading the Distributed Systems Group, TU Wien, Vienna, Austria. He is a member of the Academia Europaea: The Academy of Europe. Contact him at dustdar@dsg.tuwien.ac.at.