Comosum: An Extensible, Reconfigurable, and Fault-Tolerant IoT Platform for Digital Agriculture

Gloire Rubambiza
Cornell University
gloire@cs.cornell.edu

Shiang-Wan Chin
Cornell University
sc2983@cornell.edu

Mueed Rehman
Cornell University
mr2265@cornell.edu

Sachille Atapattu
Cornell University
sa2257@cornell.edu

José F. Martínez
Cornell University
martinez@cornell.edu

Hakim Weatherspoon
Cornell University
hweather@cs.cornell.edu

Abstract

This work is an experience with a deployed networked system for digital agriculture (or DA). Digital agriculture is the use of data-driven techniques towards a sustainable increase in farm productivity and efficiency. DA systems are expected to be overlaid on existing rural infrastructures, which are known to be less robust. While existing DA approaches partially address several infrastructure issues, challenges related to data aggregation, data analytics, and fault tolerance remain open. In this work, we present the design of Comosum, an extensible, reconfigurable, and fault-tolerant architecture of *hardware*, *software*, and *distributed cloud* abstractions to sense, analyze, and actuate on different farm types. FarmBIOS is an implementation of the Comosum architecture. We analyze FarmBIOS by leveraging various applications, deployment experiences, and network differences between urban and rural farms. This includes, for instance, an edge analytics application achieving 86% accuracy in vineyard disease detection. An eighteen-month deployment of FarmBIOS highlights Comosum's fault tolerance. It was fault tolerant to intermittent network outages that lasted for several days during many periods of the deployment. We introduce *active digital twins* to cope with the unreliability of the underlying base systems.

1 Introduction

Digital agriculture (DA) is the use of data-driven techniques towards a "sustainable intensification" [65] in farm productivity and efficiency. DA is the next generation stemming from *precision* agriculture, which is local, offline, precise application of farm inputs (e.g., water, fertilizer, etc.) [44, 57, 58]. In contrast, DA systems involve more complex data processing and communication, both on and off rural farms. DA systems are expected to be overlaid on existing rural infrastructures. However, rural infrastructures (e.g., Internet, power) are known to be less robust than urban infrastructure [17]. This is due, in part, to sparse populations [41], urban-centered technology design and standards [17], frequent outages [42], and limited maintenance [27]. These challenges make networked data aggregation and analytics on rural farms difficult [87].

While state-of-the-art approaches address several of these DA issues [26, 38, 84, 87], a lot of challenges related to data aggregation, data analytics, and fault tolerance remain open. First, although the diversity of sensor providers is growing [4, 5, 79, 82, 83, 86], data aggregation is difficult because of distributed data sources, incompatible sensors and data formats, and software dependencies. Second, the set of data analysis methods are increasingly leveraging advanced techniques such as machine-learning (ML) [45, 53, 68]; however, existing data analysis platforms rarely account for the variety of sensing mechanisms and crop types (e.g., row crop vs specialty farms). Third, state-of-the-art platforms [78, 87] partially address rural Internet and power challenges. However, fault tolerance is difficult to achieve across heterogeneous devices, networks, and cloud services. Overcoming these challenges requires extensibility, reconfigurability, and fault-tolerance in the (1) underlying sensors and networking, (2) overlaying software, and (3) supporting cloud services.

In this paper, we present Comosum,[1] a cloud-based hardware/software architecture that takes a significant step toward this goal. Comosum is designed for researchers integrating heterogeneous DA platforms. To address the heterogeneity, Comosum relies on prior strong systems concepts such as separation of devices and device drivers [73], modularity [28, 43, 61], and reconfigurability of closed-source software/hardware [18]. Our general approach to DA-enabled farms is modular with abstractions for *hardware* (i.e., sensors and networking devices), *software*, and the *distributed cloud* (i.e., a cloud that combines the edge cloud at the farm, the public cloud, and sensor vendor clouds).

Specifically, the Comosum design presents three principles:

- **Extensible modules**: Comosum modules (§§ 3.2) provide an abstraction on the acts of sensing, storing, computing, and actuating farm data. This abstraction is de-

[1]Named after Chlorophytum comosum, also known as spider plants, for their extensible leaves and adaptability to many conditions [67].

rived from the object-oriented programming idea of inheritable *instances*, which can be customized for different DA applications. Note that these modules are oblivious to the underlying hardware. This design consideration follows from the principle of dumb switches and smart control planes in Software-Defined Networking (SDN) [18]. In this manner, given a uniform API, the hardware (i.e., sensing and networking devices) and software modules can evolve independently. Following from Software-Defined Networking, Comosum is also known as the **Software-Defined Farm** (or **SDF**) [75].

- **Fault-tolerant distributed cloud**: The Comosum distributed cloud (§§ 3.3) addresses challenges in supporting elastic, vendor-neutral, and fault-tolerant data aggregation/analytics. Specifically, we highlight the complex data path where some data must be pulled directly from the sensor vendor, despite farm networking challenges. This need for networked data aggregation across the farm (or "edge") cloud, public (or "core") clouds, and private (or "vendor") clouds distinguishes the consequential fault tolerance trade-offs that are unique to DA environments (e.g., plants might die if not irrigated on time). Ultimately, by reimagining prior approaches such as CloneCloud [19], the Comosum design enables offline data collection and edge analytics during network outages.

- **Reconfigurable control plane**: Given software abstractions and distributed cloud deployment environments, the Comosum control plane (§§ 3.4) coordinates inter-module communication. To maintain reconfigurability and extensibility to heterogeneous devices, this component draws from the separation of devices and device drivers, inter-process communication (IPC), and SDN. Specifically, Comosum modules rely on a message-passing [88,91] protocol to abstract away the distributed deployment environments.

We implemented a version of Comosum which we call FarmBIOS. We have deployed multiple FarmBIOS instances in the Azure, AWS, and Google clouds. Further, we deployed these instances on one commercial farm and two research farms. Commercial farms are for-profit. In contrast, research farms are associated with land-grant universities [1,9]. Therefore, they benefit from the university's resources. As a result, we operate multiple deployments (both in open fields and greenhouses) throughout the year with different Comosum module, cloud, and control plane configurations. The configurations differ such that they can meet the needs of animal (§§ 5.1), row crop (§§ 5.3), and specialty farms (§§ 5.2).

The results based on deployments and evaluation show that Comosum supports extensible, reconfigurable, and fault-tolerant DA systems. First, we applied FarmBIOS instances to a plant water stress application and two ML-based applications yielding 86% and 97% training accuracy in vineyard and dairy cow disease detection, respectively (§ 5). Second, we present an 18-month deployment with a million sensor readings from 80 sensors networked over cutting-edge hardware in two farm edge clouds (§§ 5.3). We further analyze Comosum's reconfigurability trade-offs based on our extensive deployment experience, application requirements, and comparative spectrum measurements of a 55-acre urban farm and a 615-acre rural farm (Appendix B). Third, we show Comosum's adaptability to faulty sensors in the field (§§ 6.2), its tolerance to a network outage over multiple days (§§ 6.1) at the edge, and the edge analytics' resilience to network outages (§§ 5.2). Experience deploying Comosum had several surprising architectural implications. First, failures at the edge (i.e., in the sensors and networking) vary differently than failure towards the core of the cloud (i.e., Internet and cloud module failures), and the differences in failure scope had to be identified, escalated, and optionally tolerated or repaired. Intermittent network failures were tolerated through offline data collection at the edge, or by directly performing analytics at the edge. Long-term sensor failures were escalated by expanding the scope to their cloud-based digital twins and notifying human operators. We call this an *active* digital twin. A key takeaway was that the time to detect and need to tolerate a system component failure varied from seconds to weeks. Second, frequent unannounced API or data format changes by vendors led to many errors. Comosum evolved to shield DA application developers from this complexity by providing a uniform API; specifically, a unstructured platform layer and structured application layer. Third, Comosum was made cloud provider agnostic by providing a cloud-independent layer (i.e., through cloud-agnostic abstractions such as tables/functions) and a separate cloud-dependent layer (through cloud-dependent services such as Azure Table and AWS Simple Notification Service).

We present an experience with a deployed networked system for digital agriculture, our contributions are as follows:

- Three case studies to demonstrate DA research challenges, and how they motivate our design goals

- An integrative approach that applies and advances the state-of-the-art to a portfolio of system challenges associated with building Comosum effectively across multiple subsystems and contexts (Table 3)

- The design and implementation of Comosum, a cloud-based architecture that unifies state-of-the-art DA approaches under a single interface by distilling largely complex black-box technologies down to classical ideas

- Evidence that Comosum supports various applications, tolerates intermittent network failures, and can be deployed across different farm types and cloud providers

Paper outline: § 2 describes three DA case studies to motivate design goals. § 3 delves into the Comosum system software architecture. § 4 instantiates FarmBIOS, a Comosum implementation that unifies otherwise incompatible DA systems. § 5 describes three FarmBIOS applications and deployment contexts. § 6 describes system adaptations to ensure long-term maintenance. § 8 puts Comosum in the context of prior work before concluding in § 9. Appendix B demonstrates Comosum's reconfigurability trade-offs.

2 Challenges: Why DA is Hard

Comosum is borne out of four years of collaborative research building state-of-the-art systems to support DA. We describe the main challenges in building on existing technologies to motivate the architecture developed in § 3.

2.1 Challenge 1: Data Aggregation

Data aggregation involves pulling and processing data from a variety of sources. It is crucial in obtaining a holistic pulse on a farm's Internet of living things (IoLT). Consider DA researchers in animal science who need real-time monitoring of dairy cows to facilitate early disease detection [33, 34, 54, 72]. This requires integrating data from wearable and non-wearable cow sensors, herd management software, and manual data collection on site.

Existing wearable sensors capture behavioral, physiological, and performance parameters such as physical activity, rumination and eating time, estrous behavior, and internal temperature (e.g., [2,3]). Non-wearable sensors include cameras, weather stations, milk meters and near-infrared spectrometers, and weight scales that deliver milk yield and body weight from a static location as cows pass through with every milking session (e.g., [4,79,83]). The data monitoring is possible via various sensor provider software running on the farm computer (e.g., [5,82,86]). The sensor providers control and avail the data for download either as raw files (via FTP dumps) or JSON (via scripted API calls to the providers' servers).

However, data aggregation efforts face five major hurdles. First, the sensor data are siloed in monolithic platforms with *incompatible APIs*. Second, the datasets are delivered in various *incompatible formats* (e.g., Excel, JSON, or DIF). Third, the data is *distributed* between the farm computer and numerous sensor providers' servers. Fourth, the patchy data integration programs are *dependent* on the farm computer's operating system (e.g., to run PowerShell scripts) and disk storage layout (e.g., hardcoded, per-provider directories). Lastly, any ad hoc integration protocol is likely to introduce *data sparsity* as new sensors are incorporated and old sensors are retired. These hurdles highlight the need to aggregate sensor data from heterogeneous devices and platforms. *The system should enable data aggregation from arbitrary sensor vendor platforms, data formats, and data locations*.

2.2 Challenge 2: Data Analytics

Data analytics involves extracting actionable insights from big data. It is important in managing and predicting farm inputs and expected outputs, respectively. Consider DA researchers in plant pathology who need fast methods to detect vineyard diseases affecting grape and wine quality [7,22,68,92]. This requires detecting symptoms typically visible on the leaves.

Existing grapevine disease detection methods include molecular tests, remote sensing, and digital models. Molecular tests involve plucking diseased leaves to be analyzed in laboratories. Remote sensing and digital models combine aerial imagery from Unmanned Aerial Vehicles (UAV), vegetation indices, and machine learning (ML) techniques [45,52,53,68].

However, these existing data analytics approaches face three challenges. First, current remote sensor data cleaning and pooling processes, which are often *manual*, do not enable cross-farm analytics. Secondly, ML models have been shown to produce *contradictory analyses* depending on the choice of hyperparameters [15,20,80]. Lastly, molecular tests are *slow* (i.e., on the order of days) to yield actionable results in a setting where every second implies further disease spread. These challenges highlight the need for a reconfigurable approach for data storage and model training to enable fast iterations in any environment. *The system should enable fast plug-and-play of different analytics modules and sensing mechanisms*.

2.3 Challenge 3: Fault Detection/Tolerance

Fault tolerance involves detecting, recovering from, and optionally repairing system faults. It is important in providing timely manual or automated interventions when farm monitoring assets such as sensors have failed. Consider DA researchers in plant breeding who need to understand water status effects on plant growth by controlling for variables such as temperature, soil moisture, and CO2 levels [49,71,85]. This requires reliable sensor data collection with both short-term and long-term storage, processing, and actuation.

Existing systems (e.g., [78,87]) generally comprise sensing hubs, an Internet gateway device, and optional cloud storage and processing. The sensor hubs communicate to the gateway over various protocols such as unlicensed TV White Spaces (TVWS) [87], ZigBee [40], etc. The gateway relays the sensor data over various media (e.g., 4G/3G [40], WiFi [84]) to diverse cloud-based routing hubs (e.g., Azure IoT Hub [87], AWS IoT Core [46]) for long-term storage. With a few exceptions [26,46], prototype deployments are often outdoors [84,87].

However, fault detection and tolerance are difficult due, in part, to three challenges related to cascading failures. First, faulty sensors affect data collection. Secondly, network outages affect data storage and data processing both locally at the farm and in the cloud. Lastly, the complexity and heterogeneity of existing hardware/software systems pose significant troubleshooting issues in the field. Individually, these failures can present real consequences in farms (e.g., plants are not irrigated, animals are not fed, etc.). This highlights the importance of fault tolerance and detection in DA environments. *The system should detect, localize, tolerate and/or repair failures in sensor, network, and software components*.

2.4 Summary and Design Goals

These case studies highlight interoperability issues within state-of-the-art. These challenges map to three core system design goals:

- **Extensibility**: In addressing data aggregation (§§ 2.1), we aim to provide an *extensible interface* that can generalize sensing, analytics, and actuation across APIs, clouds, hardware, and platforms.
- **Reconfigurability**: In addressing data analytics (§§ 2.2), we aim to *allow different points in the configuration space* towards data models that can be trained and used across different networking and cloud deployment scenarios.
- **Fault Tolerance**: In addressing fault detection/tolerance (§§ 2.3), we aim to *detect and/or tolerate intermittent failures* despite the heterogeneity of hardware, farm types, and cloud services.

3 Comosum Architecture

3.1 Overview

In this section, we describe Comosum's *hardware*, *software* and *distributed cloud* - the main components in the quest to sense, analyze, and actuate on rural farms (see Figure 1).

Hardware encompasses sensing, networking, and control devices. Sensing devices produce updates based on changes in real-world conditions such as temperature, soil moisture, vegetation density, etc. Networking devices provide infrastructure support via data routing and transfer between other devices, of which routers, switches, and antennas are typical representatives. Control devices offer digital interfaces with programmable logic control (PLC) functions. While programmable, control devices are limited to firmware running on a few kilobytes of memory.

Software encompasses possible manipulations of data generated by the hardware entities. The software modules, which we describe from left to right as shown in Figure 1, offer abstractions on the acts of sensing, storing, computing, and intervening based on real-world changes. The changes are communicated via interrupt and poll mechanisms (see Table 1). The telemetry module serves as an interface to capture and reflect physical and virtual state changes. Whereas physical updates read directly from sensing devices (e.g., GPIO pins), virtual updates are third-party reports (e.g., Excel files) from vendors on proprietary sensors (see §§ 3.3). The storage module is a logical abstraction over storage structures (databases, files, etc.) and formats (Excel, CSV, etc.). The compute module captures the various networked, temporal, and spatial arrangements of compute devices (see *Cloud* below) to produce actionable results. Together, the storage and compute modules form the analytics module. Henceforth, the analytics module is used interchangeably with the storage and compute modules. Lastly, the actuation module bridges the analytics module to control devices and farm operators.

Distributed cloud encompasses ubiquitous, convenient, on-demand network access [60] to storage media and compute devices at the farm edge, the public cloud, and the private cloud (i.e., sensor vendor servers, university servers, etc.). Storage media vary from low-end USB sticks to a pool of hard disk drives in the cloud. Compute devices span low-end Raspberry Pis at the edge to high-end virtual machines (VMs).

To achieve reconfigurability and extensibility, Comosum draws inspiration from Software Defined Networks (SDNs) [18]. Here, the data plane spans the hardware and software. The control plane then is the custom configuration process of hardware components and software modules to solve individually unique DA problems (e.g., Case 1-3 in § 2). Borrowing from the object-oriented design paradigm (OOP), the fundamental Comosum unit is the *instance*.

Comosum instance modules operate in an event-driven approach with per-module processes [88]. Independent, message-passing [91] modules have two advantages. First, this enables modules to be deployed anywhere in the distributed cloud. Secondly, it simplifies reasoning about application correctness for multi-threaded modules. For instance, irrigation should be triggered in the actuation module only after a dry forecast is observed in the compute module.

To summarize, each Comosum instance is a configuration of sensor, compute, storage, and actuation modules to map and solve different sets of real-world agricultural challenges. In the following subsections, we describe in depth the extensible Comosum modules, the Comosum distributed cloud architecture and its vendor neutrality goal (§§ 3.3), and the Comosum control plane's reconfiguration capabilities (§§ 3.4).

3.2 Comosum Modules

Telemetry Module. The telemetry module is the entry point into a Comosum instance's data plane. The telemetry module requires that any interested parties (observers) are notified of new sensor readings. To that end, we design the module as an abstract class following the observer design pattern [61]. The interface comprises *register*, *notify*, *read*, and *run* operations. In managing observers, the observer argument is an abstract data type (ADT). Upon being notified, observers receive a message indicating the state change. The message is similarly an ADT, and it is used to receive new updates through an invocation of the telemetry module's *read* method. By using an ADT, the module enables chained updates where downstream modules may serve as observables (other telemetry modules) and observers (any abstraction implementing the observer pattern). In practice, sensor updates are consumed by the analytics module.

Storage Module. The storage module partly follows the classic UNIX [73] file system interface with simple *read* and *write* operations. To align with another prevalent storage model, the change feed, the module additionally supports

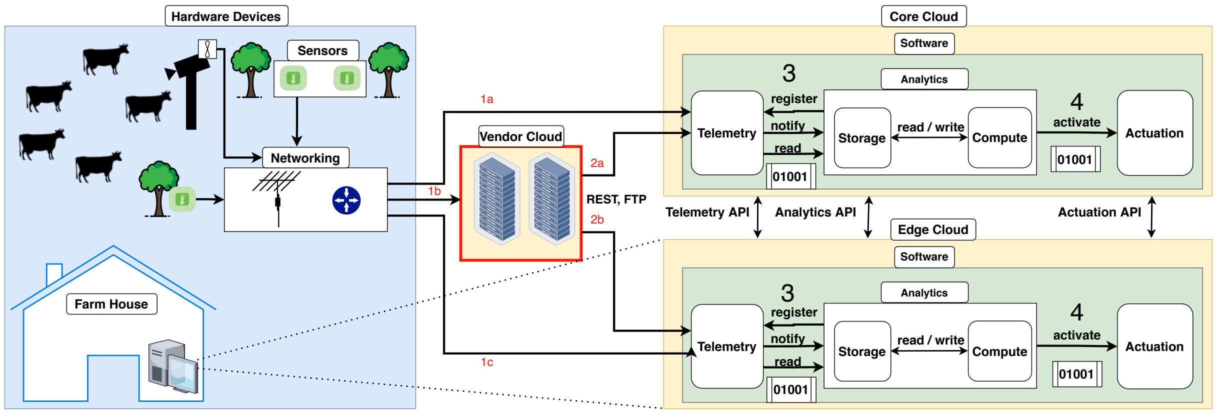


Figure 1: The Comosum distributed cloud partitions modules between the farm ("edge"), remote ("core"), and sensor vendor clouds.

change_feed and *next* methods that operate on iterators of new data (inserts, updates). Besides maintaining UNIX and database semantics, we strive for transparent access of stored objects [76]. In other words, similar to the observer argument in the telemetry module, the path argument in reads and writes is an ADT. This makes the storage module extensible to storage calls of various structures and data formats. Finally, to accommodate modules operating with topic-based storage services, the storage module supports the publish/subscribe paradigm. The pub-sub interface enables other modules to push data in addition to subscribing to updates based on topics of interest. Thus, the storage module exposes *subscribe*, *push*, *pull*, and *notify* operations.

Compute Module. The compute module operates on state changes from the telemetry and storage modules. Therefore, the module similarly follows the observer design paradigm. Acting as an observer, the compute interface supports four operations, namely *notification_sensor_rcv*, *notification_storage_rcv*, *analytics*, and *run*. The first two can be invoked to receive data based on previous compute module registrations and subscriptions to the telemetry and storage modules, respectively. The *analytics* method receives new procedure calls off the wire, and it uses the metadata and data to execute local/remote application logic. Optionally, the compute module invokes the storage module's *write* or *push* methods to store intermediary results.

Actuation Module. The actuation module is the final endpoint in the data plane. Like the sensor module, the actuation module provides an interface to physical operation of and virtual notification to real-world entities. That is, the actuation is either automated or mediated. For automated actuation, devices between the farmhouse and the field issue command messages. In this context, the module issues *activate* calls to the appropriate control devices. The command argument optionally identifies the device to execute the command. Upon an *activate* invocation in a mediated actuation scenario, the actuation module effectively is a wrapper for push notification services such as text messages and email.

3.3 The Comosum Distributed Cloud

The motivating case studies (§ 2) demonstrated that the sources of data consumed by DA applications greatly varies. While the sensor data is mostly generated locally, the initial storage and compute operations are executed on-demand by remote servers which are often owned by the sensor providers. Thus, another challenge for Comosum is achieving *vendor neutrality*. That is, hardware/software innovations must not be tied to a particular cloud-based service vendor.

To that end, we introduce the Comosum distributed cloud abstraction to support elastic, vendor-neutral sensing, storage, compute, and actuation capabilities. The Comosum distributed cloud (Figure 1) consists of the farm (or "edge"), public (or "core"), and private (or "vendor") clouds. The vendor cloud maps the more complex data path where some data must be pulled from sensor vendor servers instead of directly from the sensor abstracted away by Comosum. Unlike the "edge" and "core" clouds, Comosum modules cannot be deployed in the vendor clouds. Further, the Comosum distributed cloud design is similar in motivation to CloneCloud [19]. CloneCloud, which offloads computation from remote devices to the cloud, assumes an always-available, more powerful compute pool in the cloud. In contrast, Comosum enables computation and fault tolerance at the edge (remote device) when the (core) cloud is unavailable or too expensive to use.

While the *edge cloud*, *core cloud* and *vendor cloud* separation meets the resource elasticity and vendor neutrality goals, it also collides with limited bandwidth at farms.

On one hand, the edge cloud provides storage and computation closer to the data source. For applications with ephemeral storage and computation needs, this eliminates core cloud connectivity and latency challenges. Specifically, by leveraging networking advances such as LoRa [8] and TVWS [12], the edge cloud is capable of completely disconnected operation in deployments with large variations in area and granularity. Conversely, the edge cloud is generally unsuitable for storage- and compute-intensive Comosum applications (e.g. §§ 2.2).

On the other hand, the core and vendor clouds offer sig-

Module	Interface Method	In/Out?	Int/Poll?	Description
telemetry	register(observer)	Input	Int	Add sensor update observer
	notify(update)	Output	Int	Notify observers of a new update
	read(update)	Output	Poll	Read latest update
	run()	Input	Int	Run the sensor module
storage	write(path, data)	Input	Int	Write to storage medium
	read(path)	Output	Poll	Read from a storage medium
	change_feed()	Input	Poll	Offer an iterator to new data
	next(iterator)	Output	Poll	Get the next record from an iterator
	subscribe(subscriber, topics)	Input	Int	Register a new subscriber
	push(topic, data)	Input	Int	Publish new updates to storage
	notify(subscriber, data)	Output	Int	Push new updates to subscribers
	pull(topic)	Output	Poll	Pull recent updates, if any
	run()	Input	Int	Run the storage module
compute	notification_sensor_rcv(context)	Input	Int	Receive data from sensor update
	notification_storage_rcv(new_data)	Input	Int	Receive data from storage subscription
	analytics(arg)	Input	Int	Execute application business logic
	run()	Input	Int	Run the compute module
actuation	activate(cmd)	Output	Int	Execute a command on a control device
	run()	Input	Int	Run the actuation module

Table 1: The unified Comosum API is built from classic system design approaches (e.g., design patterns [61], UNIX file system [73]).

nificantly more storage and computation capacity, albeit at a higher network latency cost. Therefore, faced with 'reliably unreliable' [42] Internet at remote locales, an application whose progress relies on a consistent connection to the core and vendor clouds is bound to fail. Losing connection to cloud-based time critical decisions risks real consequences for farmers; plants may perish from water stress; cows may die from preventable diseases or difficult births; and vast vineyard swaths may succumb to virus infection.

In summary, the *edge cloud*, *core cloud*, and *vendor cloud* separation meets Comosum's resource elasticity and vendor neutrality goals. Note, however, that it also reveals difficult system trade-offs; for instance, latency in the context grapevine disease detection (§§ 5.3).

3.4 The Comosum Control Plane

An important Comosum goal is that initial design allows for the integration of devices and software modules. This is the guiding principle of the Comosum control plane. The control plane draws inspiration from device drivers, inter-process communication (IPC), and SDNs. Comosum leverages a diverse array of networking and storage primitives. These primitives in turn define extensible libraries and configuration templates that accommodate communication between devices and software modules from current and future DA systems.

On the hardware front, sensing devices require wrappers for new serial device drivers or wrappers to existing standard interfaces (e.g., RS485 [69]). Networking devices necessitate new packet processing interfaces. Finally, control devices typically expose wrappers for their PLCs.

On the software and cloud components, the control plane specifies inter-module communication. Due to the distributed nature of Comosum modules, we use message passing [88,91] where modules communicate as follows.

Modules call a dispatcher with a message specifying the peer module to contact. The dispatcher in turn maps message queues to socket connections to the peer modules. In this scheme, the modules remain oblivious to the underlying networking. Further, this simplifies part of the control plane to IP address tuples which can be edited as the underlying network changes. While communication between Comosum modules is sockets-based, any module calls to other systems (e.g sensor vendor clouds) uses whatever higher-level abstraction that the external systems expose (e.g., REST APIs).

As discussed in the overview (§§ 3.1), a key Comosum goal is to easily reconfigure devices and software modules to fit different applications. Therefore, Comosum implementations must remain as close to a set of reusable configuration and compilation templates as possible. Configuration templates include cloud connection strings, sensor SKUs, IP addresses, etc. Compilation templates include Docker files [24], remote procedure call (RPC) definition files, package dependencies, etc. Thus, the hardware and software components are reconfigurable to solve various DA challenges.

4 FarmBIOS: A Comosum Implementation

The particular instantiation of the Comosum architecture presented here is the Farm Basic Input Output System (FarmBIOS), drawing inspiration from its unification of routinely incompatible hardware and software systems. Henceforth, we use FarmBIOS and Comosum interchangeably. FarmBIOS code and research datasets are open source (Appendix A).

We built FarmBIOS in Python, atop Google's protocol buffers (also known as protobufs) [35]. Protobufs provide a language-independent, efficient serialization protocol that allows not only the construction of Comosum modules in numerous languages, but also extensibility of RPC templates to enable integration with arbitrary DA systems. Figure 2 shows the FarmBIOS stack. Next, we briefly describe the most salient components of FarmBIOS instances.

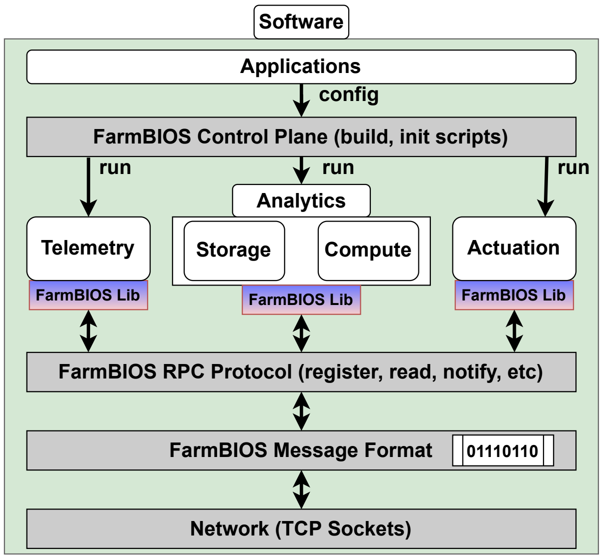


Figure 2: FarmBIOS - an implementation of Comosum

4.1 FarmBIOS Control Plane

At the top of the FarmBIOS stack, applications specify software configurations to the FarmBIOS control plane, which (1) provisions the appropriates edge and core cloud compute and storage resources (e.g., VMs, databases), (2) instantiates the required modules, and (3) builds any required Docker containers. Based on the application specific requirements, the modules are deployed and started in the edge or core cloud. Note, however, that not all modules in the middle layer are required by every application. These à la carte application configurations are at the heart of Comosum's vendor neutrality (e.g., pairing Azure compute and AWS storage).

4.2 FarmBIOS Library

The FarmBIOS library (FarmBIOS Lib) is the implementation of the Comosum module abstraction. That is, the library allows the customization of base telemetry, storage, compute, and actuation module classes to fit the purposes and configurations of different applications. Practically the library deals with challenges related to perpetual deprecation of hardware in the field and software libraries. The FarmBIOS Lib addresses this data processing challenge by providing wrappers to an array of services such as tables, databases, CSV readers, and email clients. In the current implementation, we provide wrappers around the Azure Table [62], Azure CosmosDB SQL [63], Azure Machine Learning Workspace (Azure ML) [64], Twilio [48], and OpenWeather [47] services in addition openpyxl [32] and CSV readers - with more templates to be added as our array of supported applications expands. Therefore, FarmBIOS Lib is a *structured, cloud-independent* application layer.

4.3 FarmBIOS RPC Protocol

FarmBIOS is built in a *network-agnostic* manner. This implementation choice is crucial for Comosum's extensibility to arbitrary DA platforms. Specifically, the modules are unaware of differences between local and remote peer modules. By local we mean modules operating within the same host. Local and remote operations entail modules passing and receiving messages to/from their dispatcher. The dispatcher is tasked with routing the procedure call to the appropriate peer module based on the control plane configuration. The RPCs rely on a client-server architecture built on TCP sockets wrapped by a selector operating per-connection queues, and each connection tunnels to a peer module. In both scenarios, any data communication occurs over the common FarmBIOS RPC protocol. Note that the underlying (TCP) communication protocol is known only to the dispatcher, but not the modules. Further, the RPC protocol makes no assumptions on the data formats, thereby maintaining data introspection/formatting flexibility for applications through FarmBIOS Lib.

4.4 FarmBIOS Message Format

The Comosum modules exploit OOP, UNIX, IPC, and other intuitive semantics. In practice, however, the underlying implementations addresses two major obstacles (i.e., numerous data formats and host operating system (OS) dependencies). First, the number of data formats, which represent technical compatibility negotiations between research farms and sensor providers, are both unwieldy and subject to unexpected changes [23]. Therefore, similar to the Linux file abstraction, Comosum offers a uniform, byte-addressable format for inter-module data communication - the FarmBIOS Message Format. This format is an *unstructured* platform layer. Secondly, host operating systems eventually get upgraded or lose long term support. Comosum achieves independence from the host OS by leveraging application orchestration tools such as Docker [24] and Kubernetes [29].

5 Applications & Deployment Experiences

In this section, we describe the hardware contexts, FarmBIOS usage (§ 4), and deployment contexts of the motivating challenges (§ 2) to showcase Comosum's range of applications. Specifically, the applications are different Comosum configurations that meet the needs of animal farms (§§ 5.1), specialty farms (§§ 5.2), and row crop farms (§§ 5.3).

5.1 CowsOnFitbits

Building on Challenge 1 (§§ 2.1), CowsOnFitbits is a data aggregation component supporting early disease prediction models which achieve 97% training accuracy [59]. The edge

Training	Data Location	Data Size	Compute	Storage	Runtime	Accuracy
Edge	Local	4MB	8 CPUs	256GB	27.1s	84%
Edge	Cloud	-	8 CPUs	256GB	35.6s	86%
Azure ML	Cloud	-	2 vCPUs	100GB	86.5s	86%

Table 2: Grapevine disease detection: model training runtime and accuracy for various WineGuard configurations

cloud is a farm PC (16GB RAM, 500GB storage) running the Windows 10 Enterprise OS. The OS features a Docker engine deployed on its Windows Subsystem for Linux (WSL). The edge is connected to vendor clouds and a university cloud via a 1Gbps Ethernet connection.

CowsOnFitbits leverages FarmBIOS modules (packaged as Linux containers) as follows. Sensor reports are made available by the providers via FTP dumps to the edge cloud. The telemetry module continuously polls the local disk, awaiting the FTP dumps. New reports trigger the telemetry module's *notify* function call which, in turn, notifies its local compute module. The telemetry module's *read* method is called by the compute module. The compute module aggregates sensor data from multiple streams to be stored in the private university cloud, where a module exposes a REST API for data access and queries by ML applications. In the cloud, cows are identifiable across data streams through farm-unique cow ID's. The storage calls are made to an intermediate, non-FarmBIOS module operating a Cassandra database [70].

CowsOnFitbits has been actively tracking approximately 1,500 cows in a commercial farm for nearly three years; having collected 23GB of datasets at the time of writing. The testing/deployment experience with CowsOnFitbits offers two observations. First, the unstructured platform layer (§§ 4.4) is more stable than the structured application layer (§§ 4.2). Specifically, unannounced API/format changes on the vendor side, which routinely occur every few weeks, introduce breaking changes in FarmBIOS Lib. The API breaking changes affected the API to the *vendor-specific* application layer changes (i.e., breaks). FarmBIOS tolerates these API changes by insulating itself with *vendor-independent* layers that use unstructured files to store data from the vendor along with methods that can interpret the unstructured data according to the latest vendor interface definition. The CowsOnFitbits system adapted to a recent change in approximately one day. This required minor application changes and deploying a new Docker container. Subsequent versions may benefit from Kubernetes [29], especially its rollbacks and canary deployments. Second, we observed missing/duplicate data in the core cloud due to mismatched interpretation of floating points in protobufs vs Python, missing vendor reports, unexpected signal interference between RFID tags and electrical engines for manure systems, or farm workers tripping over wires. The floating point issue was resolved through meticulous end-to-end testing of FarmBIOS modules over the course of a year. The missing report and power outage issues remain as open issues, though we propose a potential fix (see § 6, § 7).

5.2 WineGuard

Building on Challenge 2 (§§ 2.2), WineGuard is a data analytics platform for grapevine disease detection; achieving up to 86% training accuracy. WineGuard's sensor data originates from plane flights over California vineyards in September 2020 using NASA's Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) [36]. The spectroscopic sensor data is publicly accessible from the NASA cloud [66]. The AVIRIS data is merged with disease ground truth data from molecular tests on select leaves from the same period. The merged data are uploaded to the Azure Cloud for experimental retrieval.

WineGuard employs FarmBIOS modules as follows. We built a wrapper for Azure ML [64] to kickstart a reconfigurable model training pipeline. Note that the application can similarly be built using equivalent tools from other cloud providers such as Amazon SageMaker [77]. The model training configuration includes Azure subscription ID strings, the name of a pre-provisioned workspace, the location of the training data, and a text file with required Python packages. Upon issuing the compute module's *analytics* command, the configuration is deployed for training.

We deployed the WineGuard training pipeline in an edge cloud and in the Azure Cloud (see Table 2) [2]. This deployment presents several insights. First, the model accuracy is relatively stable regardless of training location, and, as expected, training at the edge with local data incurs the least runtime. Second, there is a 31% runtime overhead when fixing the location at the edge. We observed that this is due to an initial "warm-up" of the training data download from the cloud. In the best case, cloud-based training and inference are instantaneous. Otherwise, during disconnected periods, inference can be done faster and locally at the edge. Third, the satellite coordinate data were occasionally off by a few meters compared to the ground truth disease data. Without requiring manual intervention, analytics on sparse data in near real-time was necessary to correct the errors. In particular, FarmBIOS enabled the rectification of the divergence by selecting only spectroscopic bands with reliable data while still enabling the training/inference to proceed.

5.3 WaterGuard

Building on Challenge 3 (§§ 2.3), WaterGuard is a plant water stress alert system for research farms. WaterGuard is prototyped with research software and hardware provided by

[2]We tried to use a Raspberry Pi 4B (4 CPUs, 32GB storage) as the edge cloud. However, the ARM processor could not execute Docker containers built on an x86-64 architecture (the alternative edge used here) [14]. Building on an armv7 base image also failed due to end of support for distro updates.

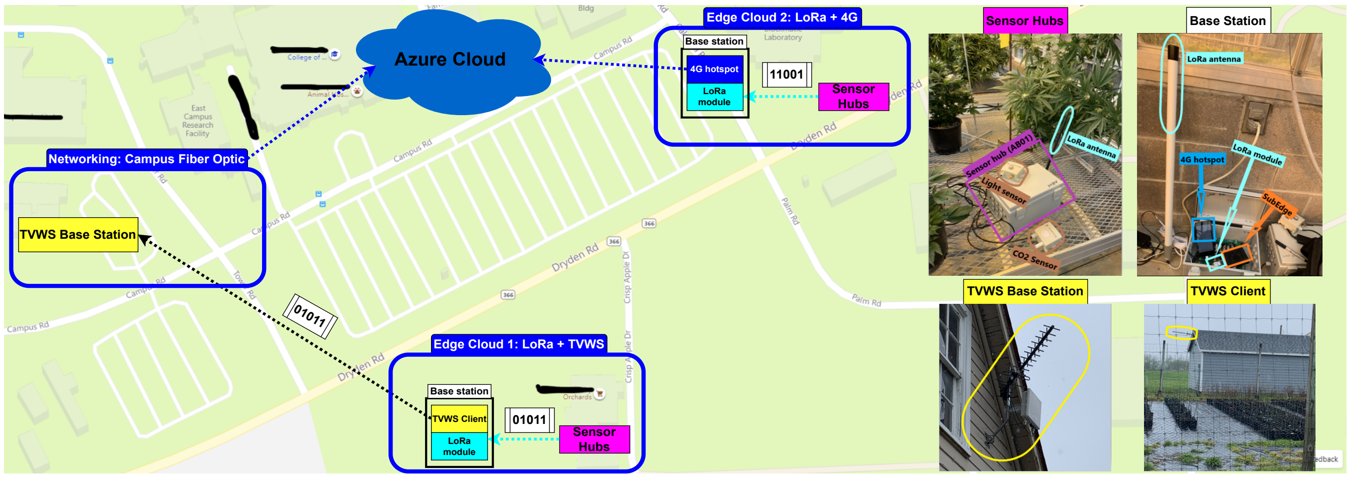


Figure 3: The Comosum hardware repertoire deployed for 18 months. Edge Cloud 1 (active since October 2020) serves an apple orchard water stress study. Edge Cloud 2 (active since March 2021) serves high-throughput corn, hemp, and strawberry breeding experiments.

Microsoft Research [87] (Figure 3). The hardware features sensor hubs from Seeed Studio operating on eight D-size batteries and supporting up to 13 analog and digital sensors. The sensor readings are networked over LoRa [8] to a base station comprised of an edge device, a LoRa module, and LoRa antenna (5dBi, 900MHz). The edge device is a general purpose UpBoard (4GB RAM, 32GB storage) running the Windows 10 IoT Core OS. In addition to the LoRa components, the device is connected to a TVWS Client (6Harmonics Inc), which provides physical and data link connectivity over a single TV channel (18, 497MHz) to a TVWS Base Station (TVBS) located at a research barn approximately a quarter of a mile away. Finally, the TVBS is wired to the university's 1Gbps fiber-optic Internet as a gateway for sensor data to Azure.

WaterGuard relies on FarmBIOS modules as follows. Sensor readings are relayed to Azure. Note that Azure table storage offers no *change feed* for observer notifications. Thus, the telemetry module relies on periodic *reads* (i.e., polling the storage module) to detect inserts that, in turn, should trigger its *notify* method. The analytics unit is notified via a *notification_sensor_rcv* call to the compute module. Next, based on the configuration received from the new update, the compute module *reads* from the shared storage module to get data on the appropriate sensor hub and start the *analytics*. Finally, upon reaching an irrigation decision, the compute module calls *activate* on the actuation module which notifies the researchers over text message using the Twilio API [48].

WaterGuard has been deployed for 18 months in two edge clouds with nine sensor hubs (Figure 3). Each sensor hub averages eight sensors and 223 days of data collection. Together, the hubs have collected over a million sensor readings. The biggest insight from this deployment was the unexpected mundane work required to adapt experimental DA systems ([87]) to new settings where failures can happen anywhere in the sensor-to-cloud continuum (see § 6).

6 Adapting to the Wild

The previous section described the hardware configurations, API usage, and deployment experiences/insights from three FarmBIOS instances. Here, we present the successes and system adaptations necessary for long-term Comosum maintenance. Though the adaptations are specific to WaterGuard, the key idea of *active digital twins* is broadly applicable.

6.1 Offline Data Collection Is Not Enough

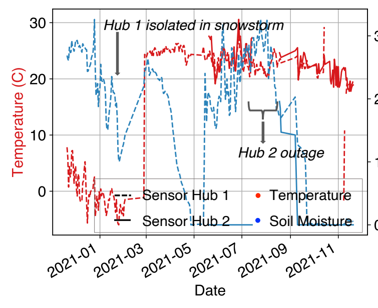
WaterGuard is capable of tolerating days-long network outages by relying on offline data collection and standard hardware redundancy. As shown in 4a, the pilot sensor hub (Sensor Hub 1) achieved disconnected operation during a snow storm and heat wave in February 2021 and May 2021, respectively. Until the 4G hotspot connectivity was restored, the sensor data was simply stored at the edge device.

However, data aggregation and analytics are still affected by faulty sensors and human error. Concretely, faulty CO2 sensors drained the batteries faster than expected (4b) and/or slight misconfigurations place data in incorrect columns. Both faults effectively result in data discontinuities (4c).

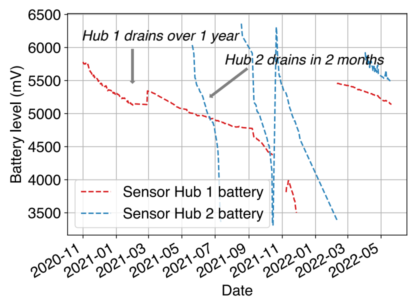
6.2 The Fix: Active Digital Twins

To streamline fault detection, Comosum evolved to include reactive monitoring [89]. That is, detecting, escalating, and optionally repairing system faults at different failure scopes. The Comosum design easily lends itself to this task by introducing ***active* digital twins.** A digital twin is a digital representation of a physical object, process, or environment that behaves like its real-world counterpart. In contrast, *active* digital twins combine the traditional digital twin concept with Comosum's actuation design.

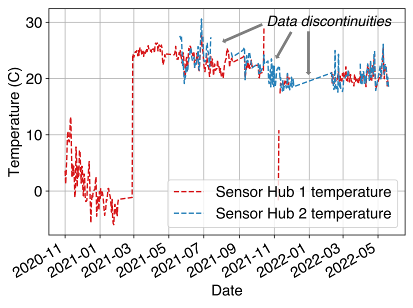
The active digital twins were an emergent concept as we iterated over FarmBIOS to make it more fault-tolerant, especially in outdoor research farm deployments where a delayed



(a) Network outages



(b) Faulty sensors



(c) Data discontinuities

Figure 4: Adapting FarmBIOS to (a) network outages, (b) faulty sensors, and (c) data discontinuities

detection could imply troubleshooting sensor hubs in two feet of hardened snow (Figure 4a, [75]). When the digital twin diverges from its physical twin beyond a reconfigurable margin (e.g., five minutes), an action is taken by the system such as sending notifications to human operators. In Comosum, only the sensor hubs are twinned. We modeled timely data collection, and divergence from the physical system is characterized by missing telemetry over a 30 minute period.

Concretely, we implemented a web-based, reconfigurable notification system with three key functions. The Status page leverages the WaterGuard sensor hubs' digital twins to display their connection state and last activity. The Configuration page provides an interface for retrieving and editing each sensor hub's port configurations. Lastly, the Notifications page provides a reconfigurable set of functions including (1) whether the notification system is **enabled**, (2) an editable **threshold** (in minutes) for triggering outage notifications, (3) an editable **frequency** for outage checks, and (4) the set of **emails** to be notified during outages. Note that the notification system relies on data aggregation and analytics across multiple, separate cloud services (i.e., Azure IoT Hub, Azure Table, AWS Simple Notification Service), and the implementation simply plugs Comosum module implementations of these services with minimal or no change to other modules. Therefore, it is broadly applicable to detect failures at different stages from the sensors at the farm to the modules in the cloud.

7 Practical Insights and Limitations

The motivating challenges (§ 2) highlighted major interoperability issues within state-of-the-art DA platforms (see Table 3 for a summary). We mapped these challenges to three core system design goals: *extensibility*, *reconfigurability*, and *fault tolerance*. We summarize below both persistent and new lessons from our experiences, and, more importantly, how FarmBIOS practically addressed the challenges. We also state the system's current limitations.

- **Extensibility to new DA vendors comes with (minor) costs**: Table 3 details the challenges in DA data aggregation. Our design goal was to provide an *extensible interface* that can generalize across APIs, clouds, hardware, and platforms. FarmBIOS provides a unified interface to merge/analyze/actuate datasets spread across the distributed cloud. For instance, CowsOnFitbits (§§ 5.1) enables the merging of datasets from four different vendors. The tradeoff is that, for each new vendor, a new script (less than 50 lines of code) must be written to move vendor reports to the appropriate directories for FarmBIOS module triggers.

- **The cloud is surprisingly unstable**: Table 3 details difficulties in system reconfigurations to support different farm networking and analytics pipelines. Our design goal was to *allow different points in the configuration space* towards data models that can be trained and used across different networking and cloud deployment scenarios. Indeed, we explored numerous hardware/software configuration possibilities (see Appendix B). However, as observed in WineGuard (§§ 5.2), the interface between the cloud and the long-lived deployed systems was not stable. In particular, the Azure ML APIs were subject to parameter deprecations which affect the WineGuard compute configurations. FarmBIOS evolved around these externalities by treating incoming parameters in telemetry and analytics modules as abstract data types. The result is a stable platform that shields users from these volatilities.

- **Failure in DA systems is the norm, not the exception**: Table 3 details the heterogeneity and failure cases that complicate DA system deployments and maintenance. Our design goal was to *detect and/or tolerate intermittent failures* despite the heterogeneity of hardware, farm types, and cloud services. In CowsOnFitbits (§§ 5.1) for instance, we observed missing data due to sensing/networking failures (frequency interference between sensors and manure systems), human factors (tripping over wires), etc. In another instance (WineGuard, (§§ 5.2)), analytics on sparse data is a necessity for deployed DA systems. We demonstrate how FarmBIOS cope with the unreliability of the underlying systems through the broadly applicable idea of *active digital*

Why It Is Hard	System Challenge	Design/Implement Decisions	Supporting Results & Contributions
Data Aggregation	Incompatible platforms	Reuse classic ideas (§§ 3.1, §§ 3.2)	Unified platform API (§ 5)
	Incompatible formats	Byte-addressable payloads (§§ 4.4)	Extensible libraries (§§ 4.2)
	Distributed data	Mask data path/sources (§§ 3.3)	Merged across clouds (§§ 5.1)
	Host dependence	Containerization (§§ 4.4)	Cross-architecture transfer (§§ 5.1)
		RPC dispatcher (§§ 4.3)	Cross-platform transfer (§§ 5.1)
Data Analytics	Manual processing	Comosum design (§ 3)	Automated processing (§§ 5.2)
	Unreproducible models	Reconfig. models (§§ 3.3, §§ 3.4)	Distributed training (§§ 5.2)
	Data sparsity	Active digital twins (§§ 6.2)	Divergence detection (§§ 5.2)
	Slow actuation	Comosum design (§ 3)	Sub-minute inference (§§ 5.2)
Fault tolerance	Faulty sensors	Standard hardware redundancy (§§ 6.1)	18-month deployment (§§ 5.3)
		Active digital twins (§§ 6.2)	Reconfig. notification system (§§ 6.2)
	Network outages	Reconfig. control plane (§§ 3.4)	Reconfig. networks (§§ 5.2, Appendix B)
		Edge analytics design (§§ 3.3)	Edge inference (§§ 5.2)
		Offline data collection (§§ 3.3)	Tolerate 7-day outage (§§ 6.1)
	Complexity/heterogeneity	Modularity (§§ 3.2)	Comosum design (§ 3)

Table 3: A summary mapping of systems challenges to Comosum design decisions and their supporting results

twins. In the case of frequency interference, for example, the RFID sensors could be twinned. In the vineyard context, the error message indicating mismatch between ground truth and satellite data could be used as input for a digital twin model.

- **FarmBIOS evolution and limitations**: As emphasized above, failure is the rule and not the exception. FarmBIOS was designed to tolerate failure, and as stated in §§ 2.3, plant and livestock depend on a robust system. Therefore, the system was improved over time. For instance, the active digital twin implementation relied on the telemetry module abstraction to increase fault tolerance. One limitation is the *lack of support for automated movement of computations between the edge and core clouds during permanent power/network outages*. Recall, however, that the edge side of the architecture is, in fact, capable of operating autonomously, at a minimum to retrieve and store sensor data (§§ 5.3) and, if so configured, perform local computation (§§ 5.2). We leave this limitation for future endeavors.

8 Related work

8.1 Programming Frameworks

To streamline application development and partitioning for resource-constrained environments, recent community efforts leverage the cloud for mobile and IoT applications. By rewriting application executables, the CloneCloud [19] architecture intelligently partitions program portions for dynamic execution between mobile devices and their cloud twins. The partitioner identifies expensive application portions through static and dynamic code analysis that informs an optimizer to solve the execution partitioning challenge. Along with EdgeProg [55] and like CloneCloud, the TinyLink [26, 39] systems form a set of cloud-native, generative systems of hardware configurations and software executables for IoT applications. TinyLink and EdgeProg expose high-level APIs and If-This-Then-That (IFTTT) languages to abstract away the low-level knowledge for developers, respectively.

In line with the prior work, Comosum exposes high-level APIs for interfacing with IoT platforms without deep knowledge of the underlying networking and hardware. Unlike CloneCloud, Comosum partitions applications at the module level, not the instruction level. Departing from EdgeProg's use of IFTTT and TinyLink's exclusive support of application development in C-like languages, Comosum supports module development with any language compatible with the (de)-serialization protocol shared by the modules.

8.2 Agricultural Sensor Networks

The rise of low-cost IoT sensor networks has led to an explosion of new communication standards and protocols being ported to industrial and consumer applications. For example, like WaterGuard, Gutiérrez *et al.* [40], Ahmad *et al.* [6], and Vasisht *et al.* [87] showcase the application of GPRS, XBEE, and TVWS technologies to agricultural monitoring systems, respectively. Further, Ayoub *et al.* [11] and Jawad *et al.* [50] present detailed overviews of both power-hungry (e.g., WiFi, Bluetooth, etc.) and low power wide area network (LPWAN) technologies (e.g., LoRa, NB-IoT, etc.) and their recent applications to, among others, dairy health care, automation, and greenhouse monitoring. Comosum demystifies these novel networking technologies' potential and limitations to interdisciplinary audiences interested in similar applications.

Besides IoT networking standards, the literature identifies open challenges in IoT networking, hardware, and software (co)design. The most salient include extensibility [81], durability [13], reliability [37, 81], modularity [81], scalability [37], energy efficiency [39, 81, 87], and interoperability among heterogeneous devices [81]. The Comosum design addresses extensibility, durability, reliability, modularity, configurability, and interoperability. Further, scalability is indirectly addressed through decoupling and thin APIs that allows independent evolution of the software modules and the underlying networking hardware, protocols, and devices.

Perhaps closest to our agricultural application of edge computing is Taneja *et al.*'s SmartHerd management system [84]. Like SmartHerd, Comosum co-opts a microservice approach, wherein the sensing, compute, storage, and actuation modules can seamlessly be placed in bandwidth rich as well as constrained environments. Further in line with SmartHerd, Comosum easily aggregates data from incompatible sensor vendors' private web servers to avoid vendor lock-in. Still, § 5 shows FarmBIOS's extensibility beyond dairy applications.

8.3 IoT Architecture Abstractions

Sisinni *et al.* [81] define a *reference IoT architecture* as a "higher level of abstraction description that helps identify issues and challenges for different application scenarios". This definition reflects the three years of exploration that resulted in the Comosum architecture. Previous architectural approaches identify sensing or perception [10, 25, 51, 90], physical [56], interface [90], networking [10, 25, 51, 90], transport [56], middleware [56], and service or application [10, 25, 51, 56, 90] layers as essential to an IoT application.

Although the architectures fundamentally serve applications with different business and technical needs, their essential layers are modules in the Comosum design. Thus, the benefit of Comosum is its partition of the physical *hardware* from the *software* that manipulates the networked data. That is, the *software*, by acting as a collection of byte-passing modules, is extensible because it is agnostic to the evolution, intricacies, protocols, or any other factors of the hardware.

9 Conclusion

In this paper, we present Comosum, a system software architecture to support digital agriculture (DA) applications in research and commercial farms. The architecture comprises *hardware*, *software*, and *distributed cloud* abstractions to build *extensible*, *reconfigurable*, and *fault-tolerant* sensor networks for farms. By supporting diverse DA applications in multiple clouds, we show that FarmBIOS, a Comosum implementation, meets these design goals. Eighteen months of Comosum instance deployments and adaptations reveal new insights on fault-tolerant sensor networks for DA. We introduce *active* digital twins to streamline fault detection, escalation, and optional repair from sensors to cloud-based modules. In ensuring that systems approaches employed in urban research farms readily map to rural farm realities, a thorough analysis highlights practical insights, limitations, and trade-offs (see Appendix) that are unique to DA applications as a starting point for community discussions of DA's potential contributions to networked system design and implementation beyond the current state-of-the-art.

Acknowledgments

This work was funded in part by a Microsoft Cornell Digital Agriculture Summer Research Fellowship, a Microsoft Investigator Fellowship, the US National Science Foundation (NSF) (Grants #1922551, #2019674, and #1955125), the U.S. Department of Agriculture (USDA) (Grants #2017-67015-26772 and #2023-77038-38865). The authors appreciate deployment maintenance and software development efforts from Kevin Huang, Yifan Zhao, and Annie Kimmel. The various SDF projects benefited from expertise and logistical support from our collaborators in Animal Science (Julio O. Giordano and Martin M. Pérez), Plant Pathology (Fernando Emiliano Romero Galván, and Kaitlin M. Gold), and Plant Science (Savanah Dale and Mike Gore) to realize the CowsOnFibits, WineGuard, and WaterGuard applications, respectively. The networking infrastructure for this project would not have been possible without the help of numerous Information Technology professionals (especially Scott Yoest, Dora Abdullah, and Avery Quinn Smith), building managers (especially Scott Albrecht), orchard managers, and carpenters at Cornell University. The authors would like to thank Ken Birman, Cody Sunderlin, Kevin Negy, the anonymous reviewers, and our shepherd, Sándor Laki, for their constructive feedback and insightful discussions during the manuscript's preparation and artifact evaluation. Lastly, the authors appreciate the support from Ranveer Chandra, Elizabeth Bruce, and Tusher Chakraborty at Microsoft for providing access to cloud credits and guidance in deploying the FarmBeats platform.

References

[1] 1890FOUNDATION, . Our History - Land Grant and Universities: A Primer. https://www.1890foundation.org/history-of-land-grant-universities.

[2] AFIMILK. AfiAct II: The Leading Cow Leg Sensor. https://www.afimilk.com/cow-monitoring#afi-collar, Nov. 2021.

[3] AFIMILK. AfiCollar: Advanced Neck Collar for Cow Monitoring. https://www.afimilk.com/cow-monitoring#afi-collar, Nov. 2021.

[4] AFIMILK. AFILAB. https://www.afimilk.com/afilab/, Nov. 2021.

[5] AFIMILK. mySilent Herdsman. https://my.silentherdsman.com/\#/login, Nov. 2021.

[6] AHMAD, N., HUSSAIN, A., ULLAH, I., AND ZAIDI, B. H. IOT based Wireless Sensor Network for Precision Agriculture. In 2019 7th International Electrical Engineering Congress (iEECON) (2019), pp. 1–4.

[7] AL-SADDIK, H., SIMON, J.-C., AND COINTAULT, F. Assessment of the optimal spectral bands for designing a sensor for vineyard disease detection: The case of 'Flavescence dorée'. Precision Agriculture 20, 2 (2019), 398–422.

[8] ALLIANCE, L. A Technical View of LoRa and LoRaWAN. LoRa Alliance (2015).

[9] APLU, A. Land-Grant University FAQ. https://www.aplu.org/about-us/history-of-aplu/what-is-a-land-grant-university/.

[10] ATZORI, L., IERA, A., AND MORABITO, G. The Internet of Things: A survey. Computer Networks 54, 15 (2010), 2787–2805.

[11] AYOUB, W., SAMHAT, A. E., NOUVEL, F., MROUE, M., AND PRÉVOTET, J.-C. Internet of Mobile Things: Overview of LoRaWAN, DASH7, and NB-IoT in LPWANs Standards and Supported Mobility. IEEE Communications Surveys Tutorials 21, 2 (2019), 1561–1581.

[12] BAHL, P., CHANDRA, R., MOSCIBRODA, T., MURTY, R., AND WELSH, M. White Space Networking with Wi-Fi like Connectivity. In Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (New York, NY, USA, Aug. 2009), SIGCOMM '09, Association for Computing Machinery, pp. 27–38.

[13] BAUER, J., AND ASCHENBRUCK, N. Design and Implementation of an Agricultural Monitoring System for Smart Farming. In 2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany) (2018), pp. 1–6.

[14] BIRADAR, P. Standard_init_linux.go:211: Exec user process caused "exec format error". https://stackoverflow.com/questions/58298774/standard-init-linux-go211-exec-user-process-caused-exec-format-error.

[15] BOUTHILLIER, X., LAURENT, C., AND VINCENT, P. Unreproducible Research is Reproducible. In Proceedings of the 36th International Conference on Machine Learning (June 2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of Proceedings of Machine Learning Research, PMLR, pp. 725–734.

[16] BROADBANDNOW. Expert Overview of HughesNet's Services. https://broadbandnow.com/HughesNet, May 2021.

[17] BURRELL, J. Thinking Relationally about Digital Inequality in Rural Regions of the US. First Monday 23, 6 (2018).

[18] CASADO, M., KOPONEN, T., SHENKER, S., AND TOOTOONCHIAN, A. Fabric: A Retrospective on Evolving SDN. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks (2012), pp. 85–90.

[19] CHUN, B.-G., IHM, S., MANIATIS, P., NAIK, M., AND PATTI, A. CloneCloud: Elastic Execution between Mobile Device and Cloud. In Proceedings of the Sixth Conference on Computer Systems (New York, NY, USA, 2011), EuroSys '11, Association for Computing Machinery, pp. 301–314.

[20] COOPER, A. F., LU, Y., FORDE, J. Z., AND SA, C. D. Hyperparameter Optimization Is Deceiving Us, and How to Stop It. In Advances in Neural Information Processing Systems (2021).

[21] DB, R. T. W. Channel Search. https://usa.wavedb.com/channelsearch/tvws, May 2021.

[22] DI GENNARO, S. F., BATTISTON, E., DI MARCO, S., FACINI, O., MATESE, A., NOCENTINI, M., PALLIOTTI, A., AND MUGNAI, L. Unmanned Aerial Vehicle (UAV)-based remote sensing to monitor grapevine leaf stripe disease within a vineyard affected by esca complex. Phytopathologia Mediterranea 55, 2 (2016), 262–275.

[23] DILORENZO, J., ZHANG, R., MENZIES, E., FISHER, K., AND FOSTER, N. Incremental Forest: A DSL for Efficiently Managing Filestores. SIGPLAN Not. 51, 10 (Oct. 2016), 252–271.

[24] DOCKER. Docker Personal: Get Started with Docker for Free. https://www.docker.com/products/personal, Nov. 2021.

[25] DOMINGO, M. C. An overview of the Internet of Things for people with disabilities. Journal of Network and Computer Applications 35, 2 (2012), 584–596.

[26] DONG, W., LI, B., GUAN, G., CHENG, Z., ZHANG, J., AND GAO, Y. TinyLink: A Holistic System for Rapid Development of IoT Applications. ACM Trans. Sen. Netw. 17, 1 (Sept. 2020).

[27] DUARTE, M. E., VIGIL-HAYES, M., ZEGURA, E., BELDING, E., MASARA, I., AND NEVAREZ, J. C. As a Squash Plant Grows: Social Textures of Sparse Internet Connectivity in Rural and Tribal Communities. ACM Transactions on Computer-Human Interaction 28, 3 (July 2021), 16:1–16:16.

[28] ENGLER, D. R., KAASHOEK, M. F., AND O'TOOLE, J. Exokernel: An operating system architecture for application-level resource management. In Proceedings of the Fifteenth ACM Symposium on Operating

Systems Principles (New York, NY, USA, Dec. 1995), SOSP '95, Association for Computing Machinery, pp. 251–266.

[29] FOUNDATION, C. N. C. Production-Grade Container Orchestration. https://kubernetes.io/, Nov. 2021.

[30] GALLARDO, R., AND WHITACRE, B. A Look at Broadband Access, Providers and Technology. Tech. rep., Purdue Center for Regional Development, West Lafayette, IN, USA, 2019.

[31] GARNETT, P., AND ROBERTS, S. Overview of Internet service provider technology considerations for rural broadband deployments. Tech. rep., Microsoft Airband Initiative Team, Redmond, WA, USA, 2018.

[32] GAZONI, E., AND CLARK, C. Openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files, Jan. 2020.

[33] GIORDANO, J., PEREZ, M., RIAL, C., NYDAM, D., YOU, Y., WANG, Y., AND WEINBERGER, K. Improving dairy cow health monitoring and management using automated sensors. In Conference of Research Workers in Animal Diseases. Abs (2021), vol. 447, p. 346.

[34] GOKUL, V., AND TADEPALLI, S. Implementation of smart infrastructure and non-invasive wearable for real time tracking and early identification of diseases in cattle farming using IoT. In 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC) (2017), pp. 469–476.

[35] GOOGLE. Protocol Buffers, Feb. 2010.

[36] GOWEY, K. Airborne Visible / Infraraded Imaging Spectromer (AVIRIS NG) DATA. https://avirisng.jpl.nasa.gov/data.html, Nov. 2020.

[37] GRGIĆ, K., ŽAGAR, D., BALEN, J., AND VLAOVIĆ, J. Internet of Things in Smart Agriculture — Possibilities and Challenges. In 2020 International Conference on Smart Systems and Technologies (SST) (2020), pp. 239–244.

[38] GUAN, G., FU, K., CHENG, Z., GAO, Y., AND DONG, W. Rapid development of IoT applications with TinyLink. In 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (May 2017), pp. 956–957.

[39] GUAN, G., LI, B., GAO, Y., ZHANG, Y., BU, J., AND DONG, W. TinyLink 2.0: Integrating Device, Cloud, and Client Development for IoT Applications. In Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (New York, NY, USA, 2020), MobiCom '20, Association for Computing Machinery.

[40] GUTIÉRREZ, J., VILLA-MEDINA, J. F., NIETO-GARIBAY, A., AND PORTA-GÁNDARA, M. Á. Automated Irrigation System Using a Wireless Sensor Network and GPRS Module. IEEE Transactions on Instrumentation and Measurement 63, 1 (Jan. 2014), 166–176.

[41] HARDY, J., WYCHE, S., AND VEINOT, T. Rural HCI Research: Definitions, Distinctions, Methods, and Opportunities. Proc. ACM Hum.-Comput. Interact. 3, CSCW (Nov. 2019).

[42] HASAN, S., BARELA, M. C., JOHNSON, M., BREWER, E., AND HEIMERL, K. Scaling Community Cellular Networks with CommunityCellularManager. In 16th ${$USENIX$}$ Symposium on Networked Systems Design and Implementation (NSDI 19) (2019), pp. 735–750.

[43] HEISER, G., UHLIG, V., AND LEVASSEUR, J. Are virtual-machine monitors microkernels done right? ACM SIGOPS Operating Systems Review 40, 1 (Jan. 2006), 95–99.

[44] HIGGINS, V., BRYANT, M., HOWELL, A., AND BATTERSBY, J. Ordering Adoption: Materiality, Knowledge and Farmer Engagement with Precision Agriculture Technologies. Journal of Rural Studies 55 (Oct. 2017), 193–202.

[45] HRUŠKA, J., ADÃO, T., PÁDUA, L., MARQUES, P., PERES, E., SOUSA, A., MORAIS, R., AND SOUSA, J. J. Deep Learning-Based Methodological Approach for Vineyard Early Disease Detection Using Hyperspectral Data. In IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium (2018), pp. 9063–9066.

[46] IMTIAZ JAYA, N., AND HOSSAIN, M. F. A Prototype Air Flow Control System for Home Automation Using MQTT Over Websocket in AWS IoT Core. In 2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC) (Oct. 2018), pp. 111–1116.

[47] INC, O. Weather API. https://openweathermap.org/api, May 2021.

[48] INC, T. Programmable SMS. https://www.twilio.com/docs/sms/quickstart/python, May 2021.

[49] Jain, P., Liu, W., Zhu, S., Melkonian, J., Pauli, D., Riha, S. J., Gore, M. A., and Stroock, A. D. A minimally disruptive method for measuring water potential in-planta using hydrogel nanoreporters. bioRxiv (2020).

[50] Jawad, H. M., Nordin, R., Gharghan, S. K., Jawad, A. M., and Ismail, M. Energy-Efficient Wireless Sensor Networks for Precision Agriculture: A Review. Sensors 17, 8 (2017).

[51] Jia, X., Feng, Q., Fan, T., and Lei, Q. RFID technology and its applications in Internet of Things (IoT). In 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet) (Apr. 2012), pp. 1282–1285.

[52] Jiménez-Brenes, F. M., López-Granados, F., Torres-Sánchez, J., Peña, J. M., Ramírez, P., Castillejo-González, I. L., and de Castro, A. I. Automatic UAV-based detection of Cynodon dactylon for site-specific vineyard management. PloS one 14, 6 (2019), e0218132.

[53] Kerkech, M., Hafiane, A., and Canals, R. Vine disease detection in UAV multispectral images using optimized image registration and deep learning segmentation approach. Computers and Electronics in Agriculture 174 (2020), 105446.

[54] Kim, H., Min, Y., and Choi, B. Real-time temperature monitoring for the early detection of mastitis in dairy cattle: Methods and case researches. Computers and Electronics in Agriculture 162 (2019), 119–125.

[55] Li, B., and Dong, W. EdgeProg: Edge-centric Programming for IoT Applications. In 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) (Nov. 2020), pp. 212–222.

[56] Liu, C. H., Yang, B., and Liu, T. Efficient naming, addressing and profile services in Internet-of-Things sensory environments. Ad Hoc Networks 18 (2014), 85–101.

[57] Lowenberg-DeBoer, J. The Precision Agriculture Revolution. Foreign Aff. 94 (2015), 105.

[58] M, A., and S, I. Effects of Precision Irrigation on Productivity and Water Use Efficiency of Alfafa under Different Irrigation Methods in Arid Climates. Journal of Applied Sciences Research 7, 3 (2011), 299–308.

[59] M. Perez, M., Yu, Y., Wang, Y., Weinberger, K. Q., Nydam, D., and O. Giordano, J. Performance of the machine learning method XG- Boost for prediction of clinical health disorders in lactating dairy cows. Journal of Dairy Science 103, 1 (June 2020), 127–127.

[60] Mell, P., and Grance, T. The NIST Definition of Cloud Computing. Tech. rep., National Institute of Standards and Technology, Gaithersburg, MD, USA, 2011.

[61] Microsoft. Observer Design Pattern. https://docs.microsoft.com/en-us/dotnet/standard/events/observer-design-pattern.

[62] Microsoft. Get started with Azure Table storage and the Azure Cosmos DB Table API using Python. https://docs.microsoft.com/en-us/azure/cosmos-db/table-storage-how-to-use-python, May 2021.

[63] Microsoft. Quickstart: Build a Python application using an Azure Cosmos DB SQL API account. https://docs.microsoft.com/en-us/azure/cosmos-db/create-sql-api-python, May 2021.

[64] Microsoft. What is an Azure Machine Learning workspace. https://docs.microsoft.com/en-us/azure/machine-learning/concept-workspace, May 2021.

[65] Mueller, N. D., Gerber, J. S., Johnston, M., Ray, D. K., Ramankutty, N., and Foley, J. A. Closing yield gaps through nutrient and water management. Nature 490, 7419 (2012), 254–257.

[66] NASA. Enabling Earth Science in the Cloud. https://earthdata.nasa.gov/esds/cloud, Jan. 2021.

[67] NCSTATE, E. Chlorophytum comosum (Anthericum Comosum, Chlorophytum, Ribbon Plant, Spider Ivy, Spiderplant, Spider Plant, Walking Anthericum) | North Carolina Extension Gardener Plant Toolbox. https://plants.ces.ncsu.edu/plants/chlorophytum-comosum/.

[68] Nguyen, C., Sagan, V., Maimaitiyiming, M., Maimaitijiang, M., Bhadra, S., and Kwasniewski, M. T. Early Detection of Plant Viral Disease Using Hyperspectral Imaging and Deep Learning. Sensors 21, 3 (2021), 742.

[69] Notes, E. RS485 - an introduction. https://www.electronics-notes.com/articles/connectivity/serial-data-communications/rs485-introduction-basics.php.

[70] Organization, A. Apache Cassandra | Apache Cassandra Documentation. https://cassandra.apache.org/_/index.html.

[71] Pagay, V., M, S., A, S. D., J, H. E., O, V., A, P., N, C. T., N, L. A., and D, S. A. A microtensionmeter

capable of measuring water potentials below -10 MPa. Lab Chip 14, 15 (2014), 2806–2817.

[72] PEREZ, M., CABRERA, E., AND GIORDANO, J. Effect of automating health monitoring on detection of health disorders and performance of lactating dairy cows. Journal of Dairy Science 102 (2019), 24.

[73] RITCHIE, D. M., AND THOMPSON, K. The UNIX Time-Sharing System. Commun. ACM 17, 7 (July 1974), 365–375.

[74] RIZZATO, F. Mobile Experience in Rural USA- An Operator Comparison. https://www.opensignal.com/2019/09/24/mobile-experience-in-rural-usa-an-operator-comparison, Sept. 2019.

[75] RUBAMBIZA, G., SENGERS, P., AND WEATHERSPOON, H. Seamless visions, seamful realities: Anticipating rural infrastructural fragility in early design of digital agriculture. In Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (2022), pp. 1–15.

[76] SANDBERG, R., GOLDBERG, D., KLEIMAN, S., WALSH, D., AND LYON, B. Design and Implementation of the Sun Network Filesystem. In Proceedings of the Summer USENIX Conference (1985), pp. 119–130.

[77] SERVICES, A. W. Amazon SageMaker. https://aws.amazon.com/sagemaker/, Nov. 2021.

[78] SERVICES, A. W. AWS IoT Greengrass: Build intelligent IoT devices faster. https://aws.amazon.com/greengrass/, Nov. 2021.

[79] SIMSHINE. Simshine Simam Alloy AS: Smart deterrence in the act. https://www.simcam.ai/simcam-alloy-1s, Nov. 2021.

[80] SINHA, K., PINEAU, J., FORDE, J., KE, R. N., AND LAROCHELLE, H. NeurIPS 2019 Reproducibility Challenge. ReScience 6, 2 (2020).

[81] SISINNI, E., SAIFULLAH, A., HAN, S., JENNEHAG, U., AND GIDLUND, M. Industrial Internet of Things: Challenges, Opportunities, and Directions. IEEE Transactions on Industrial Informatics 14, 11 (Nov. 2018), 4724–4734.

[82] SMAXTEC. BE SUCCESSFUL! WITH HEALTHY DAIRY COWS. https://smaxtec.com/en/.

[83] SUPPLY, G. T. Onset HOBO S-TMB-M006 Temperature Smart Sensor with 19.7' cable. https://www.globaltestsupply.com/product/onset-hobo-s-tmb-m006-temperature-smart-sensor, Nov. 2021.

[84] TANEJA, M., JALODIA, N., BYABAZAIRE, J., DAVY, A., AND OLARIU, C. SmartHerd management: A microservices-based fog computing–assisted IoT platform towards data-driven smart dairy farming. Software: Practice and Experience 49, 7 (2019), 1055–1078.

[85] VAN LEEUWEN, C., TRÉGOAT, O., CHONÉ, X., BOIS, B., PERNET, D., AND GAUDILLÈREJEAN-PIERRE. Vine water status is a key factor in grape ripening and vintage quality for red Bordeaux wine. How can it be assessed for vineyard management purposes? OENO One 43, 3 (Sept. 2009), 121–134.

[86] VAS. DairyComp: The World's Most Powerful Dairy Herd Management Tool. https://vas.com/dairycomp/, Nov. 2021.

[87] VASISHT, D., KAPETANOVIC, Z., WON, J., JIN, X., CHANDRA, R., SINHA, S., KAPOOR, A., SUDARSHAN, M., AND STRATMAN, S. FarmBeats: An IoT Platform for Data-Driven Agriculture. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17) (Boston, MA, Mar. 2017), USENIX Association, pp. 515–529.

[88] WELSH, M., CULLER, D., AND BREWER, E. SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (New York, NY, USA, 2001), SOSP '01, Association for Computing Machinery, pp. 230–243.

[89] WOOD, M. D., AND MARZULLO, K. The design and implementation of meta. Reliable distributed computing with the ISIS toolkit (1993), 309–327.

[90] XU, L. D., HE, W., AND LI, S. Internet of Things in Industries: A Survey. IEEE Transactions on Industrial Informatics 10, 4 (Nov. 2014), 2233–2243.

[91] YOUNG, M., TEVANIAN, A., RASHID, R., GOLUB, D., AND EPPINGER, J. The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System. SIGOPS Oper. Syst. Rev. 21, 5 (Nov. 1987), 63–76.

[92] ZARCO-TEJADA, P., POBLETE, T., CAMINO, C., GONZALEZ-DUGO, V., CALDERON, R., HORNERO, A., HERNANDEZ-CLEMENTE, R., ROMÁN-ÉCIJA, M., VELASCO-AMO, M., LANDA, B., ET AL. Divergent abiotic spectral pathways unravel pathogen stress signals across species. Nature Communications 12, 1 (2021), 1–11.

A Artifact Appendix

Abstract

The artifact documents our deployment experience and open-source efforts to build a **Software-Defined Farm** (or **SDF**), also known as the **Comosum** or **FarmBIOS** system in the present paper. Comosum is intended to provide an extensible, reconfigurable, and fault-tolerant platform for IoT data collection, processing, and actuation. On one hand, the paper covers the architecture (Comosum), the implementation (FarmBIOS), and our deployment experiences over 18 months. On the other hand, the artifact provides a standalone Docker image and a pointer to the open-source code that, together, can be used to demonstrate the instantiation of the telemetry, analytics, and actuation concepts. In particular, the artifact demonstrates the repeatability of these ideas through three applications: CowsOnFitbits, WineGuard, and WaterGuard.

Scope

In addition to inspecting the code and research datasets, the artifact can be used as a starting point for extending the FarmBIOS/Comosum platform with new cloud services and sensor vendors. The current release of the artifact is intended to validate the listed claims about the applications:

- The CowsOnFitbits application (§§ 5.1) can use the Comosum sensor module (aka telemetry module) and compute modules to aggregate data from **three** (anonymous) IoT vendors and **six** data sources.

- The WineGuard application (§§ 5.2) can use can use the Comosum compute module abstraction to train machine learning models and perform local inference with at least 75% accuracy in approximately 30 seconds.

- The WaterGuard application (§§ 5.3) demonstrates the potential of *active digital twins* in increasing the platform's fault-tolerance to failures in the path from the sensors to the cloud.

Contents

The artifact includes a Docker image named *comosum-atc-artifact-eval* and a zipped archive of the code base built from the usenix-atc23-artifact-eval branch and the b313d7e commit in the SDF GitHub repository.

Hosting

The artifact is hosted on the publicly-funded archival platform Zenodo under this unique DOI.

Requirements

- The Docker image was primarily built and tested on an X86-based system (Windows 10 Education OS, Version 22H2, OS Build 19045.2846 with WSL 2 installed to emulate a Linux-like environment, and Docker Desktop Version 20.10.10). Therefore, the image should be loadable on most Unix-like environments with Docker installed.

- In addition to the primary development environment listed above, we reproduced the results on a X86-based system (Ubuntu Linux OS 22.04.1, Docker Version 23.0.6) and an arm64-based Macbook Pro (macOS Ventura 13.3, Docker Version 23.0.5)

- Although we have successfully reproduced the results on a Mac with an Apple Silicon (M2) chip, we cannot guarantee reproducibility if the evaluation is conducted on macOS, especially the M chips which are known to have issues with Docker Desktop filesystem change notifications and port mapping/forward issues. The Comosum system extensively relies on change notifications and port forwarding.

B Understanding the Trade-offs

This section presents an exploration of network configurations for Comosum applications with two goals in mind. First, we showcase Comosum's potential reconfigurability from a 55-acre urban farm to a 615-acre rural farm. Secondly, we offer a way for interdisciplinary DA researchers to quickly establish their networking needs by assessing three factors: expected application payload frequencies (§§ B.1), network availability and throughput in urban versus rural locations (§§ B.2), and desired system latency (§§ B.3). The key observation is that the DA context has the potential for new lessons and challenges to well-established networking, storage, and application management assumptions. The Comosum experiences serve a crucial starting point for the community conversation.

B.1 Application Data Rates

Table 5 illustrates the significant range of data generation rates for the three Comosum applications; from a few bytes every six minutes to hundreds of MBs weekly. The WaterGuard total is based on a sensor hub deployment with seven sensors (see §§ 5.3). The WineGuard dataset is based on spectrometer values from a 500m*300m vineyard field. The CowsOnFitbits total is an estimation based on data from four sensor providers tracking approximately 1,500 cows on a commercial farm. The CowsOnFitbits sensor data generation varies from every 10 minutes to once daily.

Application	Payload	Frequency	LoRa (SF:12)	DSL	Satellite	4G LTE	TVWS (1x6)	TVWS (4x6)	Fiber-optic
WaterGuard	65B	6 min	0.44 sec	5.20e-4 sec	1.73e-4 sec	9.60e-5 sec	5.20e-5 sec	2.80e-6 sec	5.20e-7 sec
WineGuard	4MB	Daily	7.60 hr	32.00 sec	10.67 sec	5.93 sec	3.20 sec	0.17 sec	0.03 sec
CowsOnFitbits	17MB	Daily	1.39 days	2.30 min	45.87 sec	25.48 sec	13.76 sec	0.74 sec	0.14 sec

Table 4: Estimated cloud backup time for FarmBIOS applications under various network bottleneck scenarios.

Application	Payload	Frequency
WaterGuard	65B	6 minutes
WineGuard	4MB	Daily
CowsOnFitbits	17MB	Varies

Table 5: FarmBIOS application data rates and formats.

B.2 Achievable Network Throughputs

We consider five networking media for data transfers at a farm, namely LoRa, DSL, Satellite, 4G LTE, and TVWS. Table 6 shows the achievable data transfer rates for the different media. The LoRa settings reflect current settings from the WaterGuard sensor hubs.

The DSL throughput is included because DSL holds the largest footprint in rural housing units' Internet access [30]. The TVWS settings reflect observed throughputs in the literature [31], measured TV channel occupations (as of Sept. 2020) at a campus research farm and a more rural farm 25 miles away, and tower height-based channel availability estimations [21] for the two farms. Based on their GPS coordinates, the campus farm offers only separate, single channels while the more remote farm offers four contiguous channels.

Medium	Throughput	Internet?	Deployed?
LoRa (SF:12)	1.17 kbps	No	**Yes**
DSL	1 Mbps [30]	Yes	No
Satellite	3 Mbps [16]	Yes	No
4G LTE	5.4 Mbps [74]	Yes	**Yes**
TVWS (1x6MHz)	10 Mbps [31]	No	**Yes**
TVWS (4x6MHz)	186 Mbps [31]	No	No
Fiber-optic	1Gbps	Yes	**Yes**

Table 6: Rural uplink throughputs. SF = Spreading Factor.

B.3 Cloud Backup Bottleneck Analysis

The Comosum distributed cloud affords elastic compute and storage power. Practically, we transfer data not only to leverage more abundant compute resources for compute-intensive tasks in the cloud, but also to store the datasets for future retrospective analysis. Assuming each networking media (whether routing in the field or at the gateway) as an unreliable bottleneck in the data transfer, we compare the networking media/latency trade-offs in uploading each application's data.

Table 4 illustrates the expected latencies of different media for a given application. For instance, while a fiber-optic link in an urban farm would transmit CowsOnFitbits' 17MB of data in less than a second, the most popular Internet service in rural locations (DSL) would require three minutes. In another instance, while a research farm with one TVWS channel would route the data within 14 seconds at the edge, the four channels in the rural farm would transmit the same dataset in sub-second time.

In sum, by comparing urban and rural settings, this analysis shows the nuanced networking and storage strategies for DA applications in a distributed cloud setting. This motivates future research avenues in Comosum application migrations either as the network fails or edge resources deplete.