ERGO: A Scalable Edge Computing Architecture for Ag-IoT

Deepak Nadig, Sara El Alaoui and Byrav Ramamurthy

Dept. of Computer Science & Engineering, University of Nebraska-Lincoln

Abstract

As advanced data acquisition systems become increasingly available to heterogeneous, resource-constrained environments, intelligent decision-making and automation have become more prevalent. The agricultural Internet of things (Ag-IoT) application is representative of such an environment. Ag-IoT best represents a heterogeneous, resource-constrained environment, with its lack of broadband coverage in farms and rural areas. It is a heterogeneous network of sensors and equipment, with a high degree of data variability and a pressing need for automation. In this work, we propose ERGO, an edge computing architecture for resource-constrained Ag-IoT. We also develop Ag-IoT application APIs and the associated service infrastructure. Our proposed architecture manages the heterogeneity and dynamicity through its inherent composability and scalability. As such, it combines container orchestration systems such as Kubernetes with web-services APIs to provide scalable, edge-enabled Ag-IoT services. Our architecture and preliminary evaluations show that our scalable edge computing platform, which runs independent of cloud-backed assistance, can address critical challenges for resource-constrained, heterogeneous environments.

Architecture

Our proposed architecture is composed of an operations framework and a service framework.

Operations Framework: Our operations framework mainly deals with managing container lifecycles for various Ag-IoT services. For container orchestration, we employ K8s. To deploy application microservices on the edge computing system, we use deployments, statefulsets and persistent volumes. K8s services expose applications both internally (for inter-pod communication) and externally (for communication with the end-user). Applications are packaged as Docker containers.

Service Framework: Our service framework employs a cloud-native application microservices architecture. We design, deploy and manage each application as one or more microservices. Microservices communicate with each other through RESTful protocols provided by the container orchestration system. In the present form, our edge microservices do not communicate with the cloud. We are working on integrating a message broker to provide an asynchronous job queuing system.

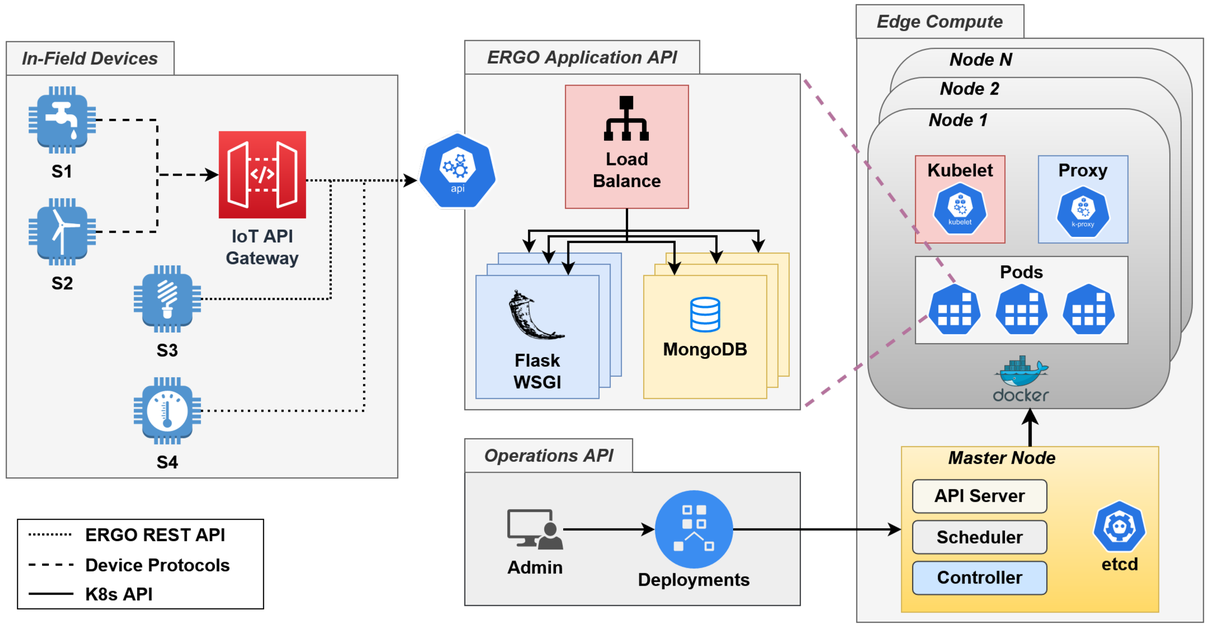

Figure 1: Proposed Edge Architecture.

Operational Challenges

Numerous challenges exist that are specific to heterogeneous Ag-IoT environments

- First, in-field sensors use a variety of protocols and wireless access technologies to communicate sensing data to endpoints. Enabling protocol conversion to TCP/IP and a well-known messaging protocol is required.
- IoT gateways to aggregate and send data from non-standard protocols must be considered.
- To include specialized hardware accelerators such as field-programmable gate arrays (FPGA) or graphical processing units (GPU), new device plugin development is necessary.
- Unlike the cloud computing systems, edge clusters for Ag-IoT are resource-constrained and have limited compute and storage resources. Therefore, caching and data storage in local databases is more conservative.
- Quality of service (QoS) classes for managing workload/resource prioritization is unavailable in K8s; our environment places more emphasis on how prioritization is achieved and how it impacts autoscaling, workload preemption/eviction and the network/traffic resources used by the workloads.

Implementation

We build a prototype system on a Raspberry Pi cluster. The cluster manages application/service deployments through the Kubernetes operations APIs. Our preliminary implementation consists of two types of services: we implement REST APIs for device management services and Ag-IoT measurement services. We implement our APIs on the Flask WSGI framework. The API documentation is also exposed using Swagger. Further, we employ response marshalling features to format, filter and render expected payload responses. We organize function-specific APIs using namespaces to allow for namespace reuse and scalability. We utilize Flask Blueprints to manage API endpoint prefixes and operations.

References

- E. A. Brewer. Kubernetes and the Path to Cloud Native. In Proceedings of the Sixth ACM Symposium on Cloud Computing, SoCC '15, page 167, New York, NY, USA, 2015. Association for Computing Machinery.
- O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. N. Hindia. An overview of Internet of Things (IoT) and data analytics in agriculture: Benefits and challenges. IEEE Internet of Things Journal, 5(5):3758–3773, October 2018.

Results

Our evaluation framework consists of a test client interacting with ERGO over a 1GbE link. We employ Apache JMeter [10] for functional load testing and performance evaluation. We present the APIs' response times and throughput performance for an increasing number of API requests in Figures 2 and 3, respectively. We test the performance of our solution by sending a fixed number of concurrent API requests for a duration of 30 seconds. We repeat the test by increasing the number of concurrent API requests per second ranging between 100 to 10,000. The average response times (Fig. 2) increase with increasing requests/second. The peak successful transactions per second increases until about 2000 requests/second and then hovers in the 550-700 transactions/second range for high request rates. We observe that our Ag-IoT application APIs scale gracefully with increasing number of requests per second. Thus, our edge computing system provides predictable performance guarantees for an increasing number of connected in-field devices and their processing requirements.

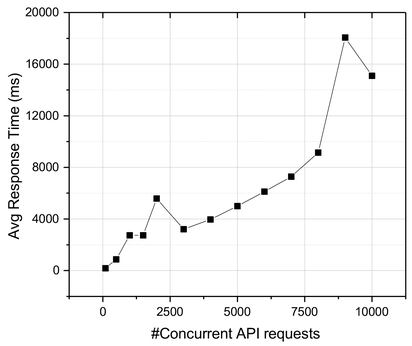

Figure 2: Latency Performance.

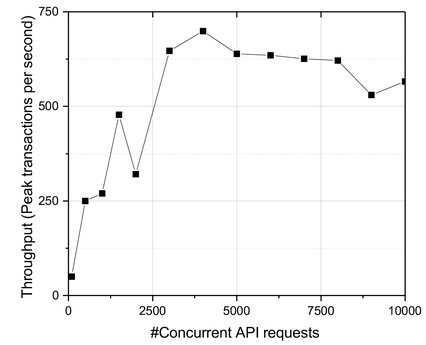

Figure 3: Throughput Performance.

QUESTIONS?

Contact us at: deepaknadig@cse.unl.edu