Demo: A System for Operating Energy-Aware Cloudlets

Thomas Rausch
TU Wien

Philipp Raith
TU Wien

Padmanabhan Pillai
Intel Labs

Schahram Dustdar
TU Wien

***Abstract*—We present an end-to-end system for operating energy-aware cloudlets with a low-footprint cluster manager and an adaptive client-side load balancing approach. Our system is designed for small-scale high-density compute clusters that host stateless services and have stringent energy resource constraints. It features cluster and service management, runtime monitoring, adaptive load balancing and cluster reconfiguration policies. Furthermore, we present an experimentation and analytics system that allows coordinated execution of complex workload experiments to evaluate different operational strategies.**

***Index Terms*—edge computing, load balancing, cloudlets**

I. Introduction

Cloudlets are a fundamental infrastructural component for edge computing [1], and it is crucial that they operate efficiently, especially in forward-deployed scenarios such as tactical environments [2]. Operating efficiently means that workload is balanced across cluster nodes as to minimize application latency, and that the cluster configuration (i.e., active nodes and request routing) is adapted during runtime to trade off latency and energy consumption.

Systems that solve similar problems in data-center scale clusters [3] are not applicable to the domain of portable energy-aware cloudlets for reasons we have outlined in a previous publication [4]. First, the operational scale impedes the use of components typical in cloud architectures, such as dedicated L4 or L7 load balancers. Second, the models used for energy management in data centers often build on assumptions that do not hold for smaller scale hardware that already has very effective built-in energy management. Third, resource management systems for cloud clusters, such as OpenStack or Kubernetes, are often very resource intensive in and of themselves and require powerful servers to operate.

This demo paper presents *Symmetry* – an end-to-end implementation of the system architecture we presented at SEC'18 [4] that addresses the described issues. The system is designed to manage high-density compute clusters with around 2-20 nodes, such as a cluster prototype we have presented in [4], or an Ubuntu Orange Box, and manage stateless services such as image recognition models, database query serving, or similar applications. Symmetry takes the role of the cluster controller, and is designed to run on a Raspberry Pi. It features service management on top of Docker, runtime monitoring of black-box metrics, power draw, and application latency, client-side load balancing, and dynamic cluster reconfiguration mechanisms. We provide load balancing and cluster reconfiguration policies such as round-robin or reactive autoscaling, but also give developers tools and APIs to build their own. Complementary, we present *Galileo* – an experimentation and analytics system for evaluating such operational policies.

II. System Description

We have described details of the system design decisions for energy-aware portable cloudlets in [4]. In this demo we showcase the implementation and its application. Figure 1 shows the overall system architecture. Symmetry orchestrates a cluster of low-power compute nodes (e.g., Xeon servers). Galileo workers are physical machines that can emulate multiple clients to generate workload. All code is open source and available in our Git repositories under the mc^2 – Mini Compute Cluster project umbrella.[1]

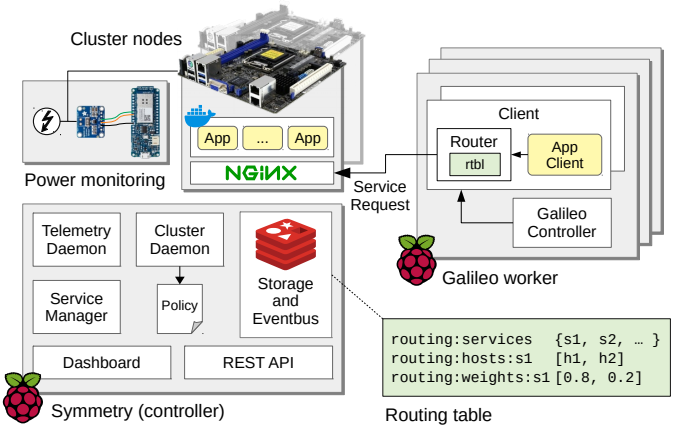


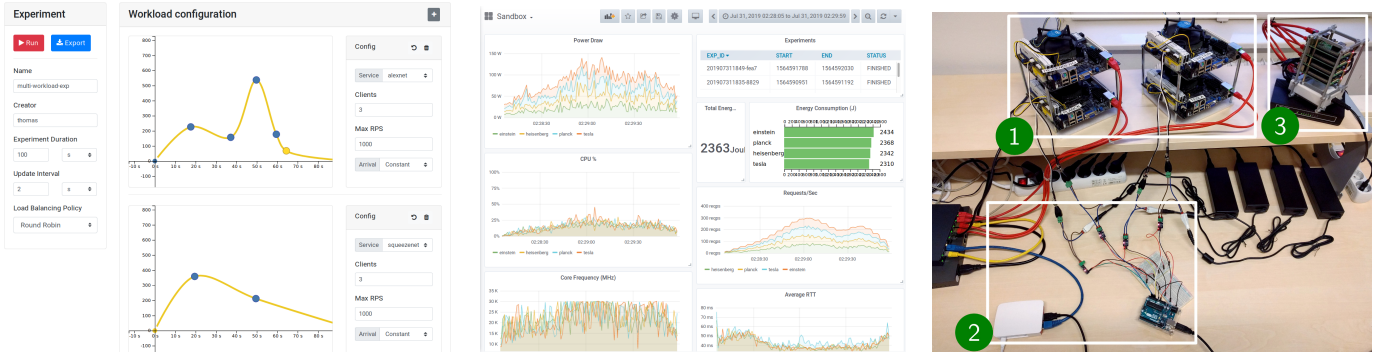
Figure 1: System architecture overview

A. Symmetry: Cluster Management and Monitoring

Symmetry comprises a set of lightweight Python components that together make up the system control software. Cluster nodes are server computers that host services, and require no software other than Docker and an SSH server.

a) Command Line Interface and REST API: Operators interact with Symmetry via a CLI to deploy services to the cluster or activate specific load-balancing policies. REST APIs provide ways to control the cluster state and request routing, and enable the modular development of additional operational logic. A runtime dashboard provides insights into the current cluster utilization, power draw, and application performance.

[1] https://git.dsg.tuwien.ac.at/mc2



(a) Experiment editor with workload profiles

(b) Galileo analytics dashboard

(c) Testbed with (1) cluster nodes, (2) Symmetry & Galileo, (3) Galileo workers

Figure 2: The Galileo subsystem enables complex experiments on physical testbeds using client and workload emulation

b) Symmetry core: The core platform component of Symmetry is a Redis instance running on the cluster controller. All layers of Symmetry, as well as the Galileo experiment platform, are integrated via this Redis instance, which facilitates both data storage (such as service metadata) as well as interprocess communication using an eventbus architecture. Due to its lightweight design and highly optimized I/O functionality, Redis performs extremely well in this scenario.

c) Service management: Applications are hosted on cluster nodes as HTTP services in Docker containers. Symmetry starts on each cluster node an NGINX instance to internally route requests to the correct container and to monitor application performance. Services are described via YAML files that specify necessary metadata. The CLI command `symmetry deploy my-service.yml` then deploys the service to each node, starts the necessary containers, registers the service endpoints, and updates the node's NGINX config accordingly.

d) Telemetry daemon: The telemetry daemon aggregates runtime metrics from various sources and publishes them as time series data into a pub/sub topic that encodes the node and the metric, for example, `telemetry:node1:cpu`. It implements pull-style monitoring by connecting to the cluster nodes via SSH and executing commands for measuring CPU utilization, CPU core frequency, or parsing the NGINX logs. For power data it connects to an Arduino that provides access to readings from Adafruit INA219 current sensors. If a node shuts down, it informs other components via the eventbus.

e) Cluster daemon: The cluster daemon enacts the load balancing and cluster reconfiguration policy. One policy that Symmetry provides out-of-the-box is *Reactive Autoscaling*, which activates or suspends a node if a system metric exceeds a given threshold for a specified amount of time. For example, our default implementation activates an additional node if the average CPU utilization is above 85% for more than 10 seconds, and suspends it if drops below 25%.

f) Client-side request routing: Making clusters appear as a single system is typically achieved via dedicated L4 or L7 load balancers, through which all service requests are routed. Instead, we take a client-side request routing approach that uses simple weighted-random load balancing, where weights are updated dynamically by the load balancing policy in a way that meets some operational goal. A routing table specifies for each service how much of the workload should be directed to a given node. Updates to the routing table are propagated to the clients via Redis pub/sub. This way, request routing is decentralized, but simplified such that the client components necessary to call services are kept simple.

B. Galileo: Experimentation and Analytics Subsystem

The main purpose of Galileo is to simplify the development and evaluation of different operational strategies implemented in Symmetry. It coordinates physical devices to emulate client workload, and provides user-facing components to define experiments and analyze the results. Galileo records monitoring data coming from Symmetry into a configurable datastore, in our case a MySQL instance. Figure 2 shows the frontend components of Galileo, and the testbed we use for the demo.

A user can interact with the experiment platform either via the experiment editor shown in 2a, or an interactive experiment shell that provides commands to scale up/down the number of emulated clients, change the load they are generating, change the balancing policy, and control the node states via Symmetry. The shell also acts a scripting environment and provides simple flow control via sleep commands. Finally, the analytics dashboard shown in Figure 2b allows ad-hoc exploration of experiment result data. It shows system metrics for each node, aggregated energy consumption, and application performance such as processing time and queuing delay.

The environment shown in Figure 2c hosts the presented system and encompasses (i) the cluster infrastructure (low-power Xeon servers), (ii) a Raspberry Pi that hosts Symmetry and the Galileo controller, and (iii) Galileo workers for emulating clients and workload.

ACKNOWLEDGMENT

This work is supported by the Austrian infrastructure program (HRSM 2016) as part of the CPS/IoT Ecosystem project. We thank Silvio Vasiljevic and Jacob Palecek, who have contributed code to the presented project.

REFERENCES

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[2] G. Lewis, S. Echeverria, S. Simanta, B. Bradshaw, and J. Root, "Tactical cloudlets: Moving cloud computing to the edge," in *2014 IEEE Military Communications Conference*. IEEE, oct 2014, pp. 1440–1446.

[3] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "Autoscale: Dynamic, robust capacity management for multi-tier data centers," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 4, p. 14, 2012.

[4] T. Rausch, C. Avasalcai, and S. Dustdar, "Portable energy-aware cluster-based edge computers," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 260–272.