A Programming Environment for Heterogeneous Stream Analytics

Roshan Bharath Das, Marc X. Makkes, Alexandru Uta, Lin Wang, Henri Bal
Vrije Universiteit Amsterdam
{r.bharathdas, m.x.makkes, a.uta, lin.wang, h.e.bal}@vu.nl

I. Introduction

Sensor-based applications using Big Data are of increasing importance in various fields. A typical example of such use cases is building health-care applications [1], [2]. In this case, a smartwatch measures the patient's heart rate. A smartphone can then analyze the gathered data and identify patterns in the patient's heart rate. However, if the data analysis is too complex to be performed on a smartphone, the computation could be offloaded to a nearby cloudlet or a remote cloud. The analysis is usually followed by a decision, and an actuation is performed accordingly (e.g., a message is sent to either the patient or the doctor). Developing such an application is intrinsically complex, as the programmer needs to reconcile different APIs specific to different platforms.

To reduce development complexity, we advocate a framework with a unified programming model for stream analytics on low power devices. Such a framework helps programmers by providing a set of unified APIs that are easy to use. We believe that a *sensing-processing-actuating* model is sufficient to build the majority of sensor-based applications that are based on stream analytics.

In this paper, we propose a unified programming environment for building sensor-based applications that can perform sensing-, processing-, and actuation-tasks in heterogeneous environments. Such devices include the wearables, phones, and edge or cloud nodes. For this purpose, we extend our Cowbird [3] framework to add support for sensing on wearables and enable actuation on both phones and wearables. We also extend our domain specific language SWAN-Song [4] to support both local and remote actuation.

II. extended Cowbird Framework

Here, we discuss the architecture and the programming abstraction of the extended Cowbird framework. It aims at helping developers to build context-aware applications in a heterogeneous environment. We focus on improving the programming environment for small footprint edge devices (such as smartphones, Rasberry Pis, smartwatches) by providing simple (less expressive) functionality at the edge. Our work can be combined with complex analytic systems like Storm [5] and Spark [6] to provide analytics across the whole IoT ecosystem.

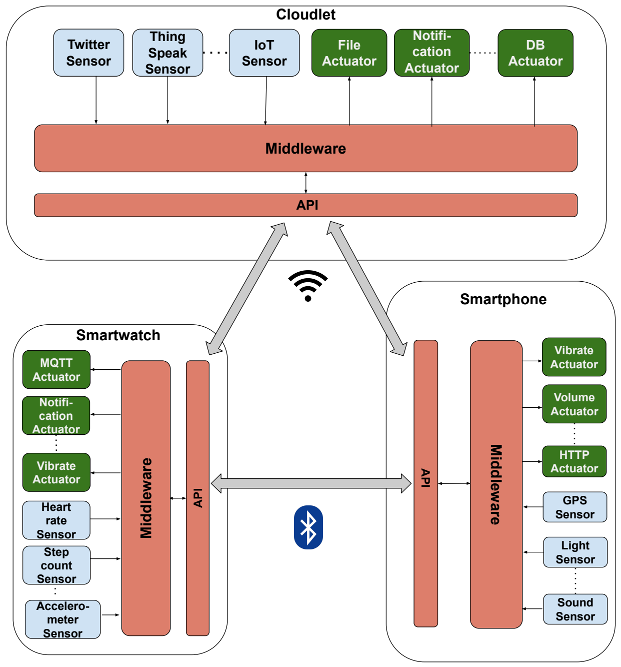

Fig. 1: An overview of the extended Cowbird architecture. (Sensors are light blue, actuators are dark green.)

A. Architecture

Figure 1 shows the architecture of the extended Cowbird framework. It contains three different types of nodes, the watch, the phone, and the cloudlet. The watch and the phone interact with each other using Bluetooth communication. Both the watch and the phone interact with the cloudlet using WiFi or 4G communication. In a heterogeneous environment, every node can perform sensing, processing, actuation, or a combination of it. We note that the watch, the phone, and the cloudlet can perform all three functions: sensing, processing, and actuation. Also, multiple watches and phones can interact with each other via the cloudlet. Currently, the framework supports both Android and Java-based IoT devices.

Next, we describe the components of the extended Cowbird framework.

1) Sensors: Sensors generate continuous data that needs to be processed. Sensors can be either hardware (e.g., GPS, accelerometer, gyroscope) or software (e.g., weather, twitter, news, stock). In the case of software sensors, the data may be generated locally or gathered remotely. Different types of sen-

TABLE I: Middleware support for sensor-based applications. P implies partial support and X implies not supported.

Characteristics	**Edgent [7]**	**Sentio [8]**	**Flink [9]**	**Cowbird [3]**
Easy sensor access	X	✓	X	✓
Perform actuation	X	X	X	X
Support for offload stream processing	X	X	✓	✓
Use of domain specific language	X	X	✓	✓
Works with low-power, heterogeneous devices	P	✓	X	P
Open source	✓	X	✓	✓

sors generate data at different frequencies. Some sensors also allow different frequencies. For example, an accelerometer sensor can generate data at four different levels of frequencies where the fastest is used for gaming purposes and the slowest for screen orientation changes.

2) *Actuators:* The sensor data is evaluated, and the result is sent to the application for further action. The actuation can be hardware-based such as vibrate, turn on the flashlight or software-based (e.g., send notifications, log, make HTTP requests). For sending the evaluation result from the phone to a server, actuators such as HTTP or MQTT [1] are used. The result received at the server as a part of actuation from the phone can be used as sensor data for further processing. In this way, we can enable data flow between nodes.

3) *Middleware:* The middleware is responsible for four main tasks: a) gathering relevant sensor data from various sensors b) processing the gathered sensor data based on a given window size, frequency and operation c) performing both local and remote actuation d) inter-device communication such as Bluetooth communication between the watch and the phone and WiFi communication between watch or phone to the cloudlet.

4) *API:* The extended Cowbird API handles incoming requests from other devices or third-party applications. Using this API, developers can subscribe to sensor data, trigger actions, or get the list of sensors and actuators available on a device.

Some preliminary functionalities for building such a framework have already been provided by existing solutions, as shown in Table I. Our existing Cowbird [3] framework already runs on resource-constrained devices, and provides support for distributed sensing, offloads computation from phone to cloud, performs stream analytics on the cloudlet, preserves privacy in the cloud [10], and makes use of a domain specific language called Swan-song [4]. We extend it to support both local and remote actuation on watches, phones and cloudlets.

B. Programming Abstraction

The extended Cowbird framework allows application developers to easily interact with various sensors and actuators. The developers can register expressions written using an extension of Swan-song to perform actuation locally and remotely.

Out of all possible types of scenarios, the paper focuses on two representative scenarios that include local and remote sensing, processing, and actuation. The first scenario LLM performs the sensing and processing on the $deviceL$ and the actuation on the $deviceM$ and the expression for this scenario is written as:

```
deviceL@sensor:path{operationX,timeWindowY}
THEN
deviceM@actuate:path
```

where the sensing expression and the actuation expression are separated by $THEN$, $deviceL$ represents the location of sensing and $deviceM$ represents the location of actuation, $sensor : path$ implies the type of the sensor and the value path (e.g., location:latitude), $operationX$ represents the type of operation (e.g., MEAN, MAX) and $timeWindowY$ represents the time window (e.g., 10s for 10 seconds), $actuate : path$ represents the type of the actuation and the value path (e.g., vibrator:vibrate). The expression computes the operation over a time window on the data generated by the sensor on the $deviceL$, and the result is sent to the $deviceM$ for actuation.

The second scenario is LMN where the first expression only gathers data from the sensor on the $deviceL$ and sends it to the $deviceM$ without doing processing (also called as ANY processing) and in the second expression the $deviceL$ gathers the actuated data as a sensor and performs processing. The result is sent to the $deviceM$ for actuation. The expressions are written as:

```
/* Expression registered on deviceL */
deviceL@sensor:path{ANY,0s}
THEN
deviceM@sensorA:pathE
--------------------------------
/* Expression registered on deviceM */
deviceM@sensorA:pathE{operationX,timeWindowY}
THEN
deviceN@actuate:path
```

C. Application

We built two applications [11] using the extended Cowbird framework: 1) An elderly care app that measures the average heart rate from the watch of an elderly person and displays it on the care taker's phone. 2) An environmental monitoring app that measures the average light from multiple watches and sends it to the cloudlet for further analysis.

III. Related Work

Various solutions exist that can help in building context-aware applications. IFTTT [12] uses trigger action-programming to build context-based recipes. They do not focus on building recipes based on sensor data. Tasker [13] allows adding rules through a graphical interface. However, they are not suitable for use as middleware for building complex sensor-based applications.

There are several stream processing platforms [6], [5] built for distributed processing on large clusters. They focus on processing data collected from smartphones and wearables in the cloud. However, they are not suitable for some scenarios that require processing locally to save energy. Alternatively, Edgent [7] is built to run on low-end devices. However, there

[1]https://mqtt.org/

is no domain-specific language to build applications easily. Some frameworks have been proposed to build context-based applications. Sentio [8] focus on virtualization of sensors and runs as middleware on the watch, the phone, and the cloud. They do not provide programming support for offloading the processing in a heterogeneous environment.

Our solution is different in that we enable programmers to build context-aware applications in a heterogeneous environment that contains different types of nodes.

IV. Conclusion

We designed the extended Cowbird framework that allows developers to build context-aware applications in a heterogeneous environment using a unified programming API that follows a sensing-processing-actuating model. The extended Cowbird framework reduces the application development complexity by allowing local and remote sensing, processing, and actuation on different categories of devices, such as the watch, the phone, and the cloudlet.

Acknowledgment

This work is funded by the municipality of Alkmaar, The Netherlands. It takes place in the context of Data Science Alkmaar led by Frans Feldberg.

References

[1] K. Sakuma, A. Abrami, G. Blumrosen, S. Lukashov, R. Narayanan, J. W. Ligman, V. Caggiano, and S. J. Heisig, “Wearable nail deformation sensing for behavioral and biomechanical monitoring and human-computer interaction,” *Scientific reports*, vol. 8, no. 1, p. 18031, 2018.

[2] F. Firouzi, A. M. Rahmani, K. Mankodiya, M. Badaroglu, G. V. Merrett, P. Wong, and B. Farahani, “Internet-of-things and big data for smarter healthcare: from device to architecture, applications and analytics,” 2018.

[3] R. B. Das, N. V. Bozdog, and H. Bal, “Cowbird: A flexible cloud-based framework for combining smartphone sensors and iot,” in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2017 5th IEEE International Conference on*. IEEE, 2017, pp. 1–8.

[4] N. Palmer, R. Kemp, T. Kielmann, and H. Bal, “Swan-song: A flexible context expression language for smartphones,” in *Third Workshop on Sensing Applications on Mobile Phones*. ACM, 2012, p. 12.

[5] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, “Storm@ twitter,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 147–156.

[6] M. Zaharia *et al.*, “Apache spark: a unified engine for big data processing,” *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[7] T. A. S. Foundation, “Apache edgent,” 2016. [Online]. Available: http://edgent.apache.org/

[8] H. Debnath, N. Gehani, X. Ding, R. Curtmola, and C. Borcea, “Sentio: Distributed sensor virtualization for mobile apps,” in *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2018, pp. 1–9.

[9] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, “Apache flink: Stream and batch processing in a single engine,” *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[10] M. X. Makkes, A. Uta, R. B. Das, V. N. Bozdog, and H. Bal, “P^2-swan: Real-time privacy preserving computation for iot ecosystems,” in *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*. IEEE, 2017, pp. 1–10.

[11] R. B. Das, M. X. Makkes, A. Uta, L. Wang, and H. Bal, “Aves: A decision engine for energy-efficient stream analytics across low-power devices,” in *2019 IEEE International Conference on Big Data*. IEEE, 2019.

[12] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman, “Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 3227–3231.

[13] Kitxoo, “Tasker for android,” 2019. [Online]. Available: https://tasker.joaoapps.com/index.html