Karibu: A Flexible, Highly-Available, and Scalable Architecture for Urban Data Collection

Henrik Bærbak Christensen, Henrik Blunck, Niels Olof Bouvin,
Robert S. Brewer, Markus Wüstenberg
Aarhus University
Department of Computer Science
{hbc,blunck,bouvin,rbrewer,markus}@cs.au.dk

ABSTRACT

Collecting data in a reliable and scalable way from a broad variety of Internet of Things (IoT) sensors requires significant effort, leading many researchers to use ad hoc data collection and storage systems that are unable to handle the quantity of data generated in dense urban environments, or to support long-term studies. In this paper, we present Karibu: a data collection architecture and reference implementation designed to meet the needs of the urban IoT community. Karibu enables sensor data collection from diverse sources and reliably stores the data in a scalable backend subsystem, allowing users to more efficiently pursue their research goals. We lay out the guiding principles of the Karibu architecture, including the data integrity and the modifiability design principles. We then detail our open source reference implementation of the architecture, which includes flexible and configurable client-side modules supporting the collection of data from heterogeneous sources and data types. Finally, we present our experiences using Karibu in two urban research projects: trip and transportation mode detection on smartphones, and exploring resource consumption in a highly-instrumented college dormitory.

Keywords

Internet of Things, sensor data, software architecture, open source, scalable, high availability, behavior change

Categories and Subject Descriptors

D.2.11 [**Software Architecture**]: Domain-specific architectures

1. INTRODUCTION

Research on the application of Internet of Things (IoT) in urban spaces requires researchers to collect, store, and analyze large quantities of data from pervasive, mobile, and heterogeneous devices. The avalanche of data from these sensors requires researchers to devote time to data management, instead of their research questions. Hard-won data may be entrusted to ad hoc systems that are not able to scale up as the project grows, or crucial data may be lost due to backend down time. We have experienced these data problems first-hand in our urban IoT research, including a project that monitors residents resource use in a highly-instrumented dormitory (the "dormitory" project), and a project that tracks users' transportation habits through sensors in their smartphones (the "mobility" cases).

Based on our research in urban IoT, a sensor data management system must: 1) support heterogeneous data from an evolving set of heterogeneous sensors, and 2) provide strong guarantees that sensed data is not lost in collection or analysis.

Unfortunately, we did not find any system that met all of these requirements for sensor data collection. Rather than develop a one-off, internal solution, we have developed an architecture for urban IoT data collection and storage we call Karibu, which means "welcome" in Swahili. A key point in its architecture is its ability to support heterogeneous types of clients, data types, and transmission formats, and 'welcome' new ones at run-time without service interruption. Besides flexibility, Karibu is designed for high availability and performance through redundant components and horizontal scaling. The integrity of received data is ensured through an event-oriented storage strategy. We have also developed an open-source reference implementation of the Karibu architecture.

The main contributions of our research are: 1) the articulation of the Karibu architectural design principles and resulting architecture, 2) a reference implementation that instantiates it, and 3) our experiences using the system for urban IoT research.

The paper begins with a review of related work on urban sensor data collection. We then describe the Karibu architecture and reference implementation, followed by our experiences using the system. We end with a discussion of the limitations of Karibu, and future work.

2. RELATED WORK

The easy availability of ubiquitous sensors, whether embedded in our surroundings, or with us in our mobile devices, has enabled pervasive sensing on a hitherto unseen scale.

Xively is a commercial public cloud service for IoT development.[1] It provides an API for client development allowing upload to and queries from their cloud storage, as well as web based monitoring and deployment. Xively seems to have been driven by requirements similar to our own, notably scalability and flexibility in data types, and indeed has similarities in its architecture (e.g., the use of a messaging system combined with time-series archiving). For IoT research, open source software such as Karibu has the advantage that it can be controlled fully by researchers themselves. Furthermore, an open source framework enables flexibility and extensibility that third party commercial services such as Xively cannot necessarily provide. Finally, for legal reasons such as confidentiality and privacy of user data, self-hosting may be preferable.

The PEIR system [6] supports sensor data collection from Symbian

[1] `http://xively.com`

Urb-IoT 2014, October 27-28, Rome, Italy

DOI 10.4108/icst.urb-iot.2014.257253

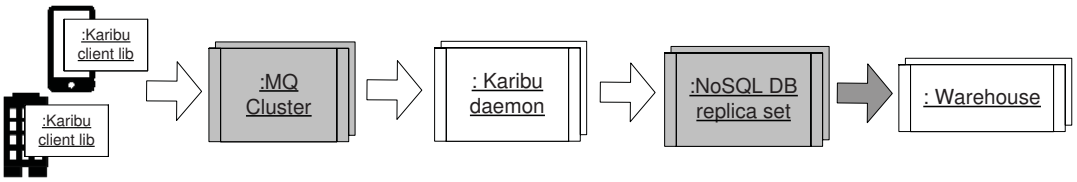

Figure 1: Karibu configured for a production environment. Gray boxes represent third party applications/processes while the white boxes are Karibu processes (using adapted UML object-diagram notation). Arrows represent data processing flows: white arrows are real-time data processing, while the dark gray one is batch processing.

and Windows phones. Collected location trace data is transmitted in bulk to a location data multiplexing web service, which forwards data to both testing and production servers, and archives an additional copy.

The DeviceAnalyzer[2] project collects usage statistics from Android smartphones, with almost 18,000 users globally. Wagner et al. [8] discuss several aspects of the project, including experiences with the data collection from heterogeneous Android devices: fault-tolerance of transmissions, reliable interpretation of timestamps, and the energy-efficiency of the app on the phone.

Some sensor data collection systems are specific to a certain domain: e.g., the WattDepot [4] system provides an ecosystem for the collection, storage, analysis, and visualization of electricity data.

With the future development and proliferation of mobile, sensing devices in mind, it becomes important to consider the robustness and longevity of the architecture and tools underpinning the research being done.

3. KARIBU SOFTWARE ARCHITECTURE

To elicit the architectural requirements, we conducted quality attribute workshops for several of our research projects. These lead to four central architectural quality attributes to support: *data integrity, modifiability, availability,* and *performance*. We translated these architectural requirements into four design principles that have guided the architecture of Karibu.

The first principle concerns data integrity, and is inspired by Marz [5], who defines a *data system*:

> A system that manages the *storage* and *querying* of data with a lifetime measured in *years* encompassing every *version* of the application to ever exist, every *hardware failure*, and every *human mistake* ever made

This definition is true for a ubiquitous data collection framework: numerous types of devices and data producers are involved, and what we "look for" in research projects will definitely vary as our understanding of the problem domain improves, meaning that the set of relevant queries on the data set will evolve over time. Marz expresses the architectural tenet for data integrity:

> As long as an error doesn't *lose or corrupt good data*, you can *fix* what went wrong [5]

This tenet leads to Karibu's first design principle:

Data Integrity Design Principle The primary storage format is event-based and minimally different from the data collection format. Raw data can only be written, never updated nor deleted. Querying is deferred to a separate (per-project) data warehouse generated by post-processing raw data.

That is, each data collection event should lead to one new document in the database, and in the same (or minimally processed) format as the raw data.

[2] `http://deviceanalyzer.cl.cam.ac.uk/`

Second, in research projects, new types and formats of data will appear after a data collection system has gone into production, and thus the system needs to support adding new data types dynamically.

Modifiability Design Principle Karibu can be configured for new data types, versions, and transmission or storage formats at run-time.

Third, Karibu is required to have high availability, i.e., to accept data from all data collection devices at any time, which leads to the:

Availability Design Principle All backtier run-time components must be redundant and fault tolerant. All storage components must be replicated.

Finally, research projects may develop new usage scenarios over time that increase the workload, which leads to the final:

Performance Design Principle Performance, high throughput, and high storage demand are achieved through horizontal scaling.

The *Karibu run-time architecture* embodies the four principles and is shown in Figure 1. Starting from the left, smartphones and other data producers have the *Karibu client library* deployed. The client is configured with a list of receiving messaging systems that form a redundant cluster. Collected data is put into a message by the client, and sent to the messaging system using the Karibu library's send(data) method, using SSL-encrypted connections for security and privacy. A number of *Karibu daemons* continuously fetch messages from the messaging systems, perform minimal processing into a suitable storage format, and finally store data in a replicated primary database system. For visualization and advanced queries, it is often useful to extract and convert data from the primary database system into a data warehouse where the resulting data is stored in a format optimized for the relevant queries.

The Karibu architecture uses a messaging system as it is a well established method to decouple the producer and the consumer of data, and is used in many high-throughput deployments, where messages must be consumed, but need not be replied to. Furthermore, Karibu uses NoSQL databases that have become common choices for data-intensive applications, where the guarantee of complete consistency of conventional databases has been traded for eventual consistency and very high scalability.

In our open-source *Karibu reference implementation*, we use RabbitMQ [7] instances configured as a cluster with mirrored and persistent queues. As long as just one of the instances is running, clients will be able to deliver data. The Karibu client libraries automatically connect to the first live messaging system in the configuration list.

The Karibu daemons consume messages round-robin from the message queue, thus automatically balancing the load. If a daemon fails, the others take over the processing of the queued messages, thus providing redundancy and fault tolerance. If all daemons fail, the messaging system will queue all incoming data until new daemons are started. The daemons perform a minimal conversion of the message into the storage format of the database. In the reference implementation, we use the NoSQL database MongoDB [1] configured as a replica set, i.e., three identical nodes that run in a master/slave setup.

Finally, the primary database is generally unsuited for complex querying due to the data integrity design principle's requirement of event-based storage. Therefore, many applications may process the raw data into secondary data warehouses so data is more readily queryable.

This setup has a high resilience against hardware failures in nodes and network; and moreover, it allows maintenance such as adding new data types and installing security updates to be performed on all machines without any service interruption.

Our reference implementation has been developed to allow it to easily be configured for development and testing, using a single message RabbitMQ broker, a single daemon, a single MongoDB instance, and running without SSL and credentials. For automated testing, all distribution can also be simulated using in-memory components.

As system load increases, all components can be horizontally scaled to increase capacity. One can add more RabbitMQ nodes (and configure the clients to use their own set of nodes for load distribution), add more Karibu daemons to increase throughput and fault tolerance, and, finally, MongoDB has strong support for automatic sharding allowing seamless addition of more nodes for increased capacity and throughput.

4. EXPERIENCE USING KARIBU

Our use the Karibu system over the last 13 months indicates that is it a viable architecture for data collection: at present its availability is at 99.85%, and our MongoDB stores more than 140 million documents. More than six new projects and data formats have been added since the initial deployment without any service interruption. The present steady-state data inflow is around 65 kb/s, while peaking at 2.1 Mb/s during a burst lasting 51 minutes.

We have used Karibu to investigate people's mobility behavior in urban areas, using smartphone apps to track their transportation events by collecting accelerometer and GPS sensor data, which is then sent to Karibu. We can analyze this data to determine how long the trip was and the mode of transportation used: walking, biking, driving, etc. The analyses can be used to inform users about their personal transportation habits, and to obtain a better overall understanding of transportation in an urban area. For example, one app informs users on whether their typical driving patterns are compatible with the more limited range of an electric car.

Karibu enables us to reliably collect data from a broad variety of user devices, which is critical due to the wide variation in devices (OS version, sensor abilities, etc.) [2]. This capability allowed us to collect *comparable* data over several months, unlike the experiences of some other systems like PIER, which were only able to compare data over a few weeks due to data collection issues [6].

Another of our research projects involves the Grundfos Dormitory Lab, a "living laboratory", where each of the 159 apartments has sensors installed that monitor indoor climate, electricity, heating, and water use.

We use the sensor data in a new methodology called Computational Environmental Ethnography, combining collective sensing and anthropological inquiries [3], to gain insight into the resource usage patterns of the residents in the dorm. Karibu's robust handling of sensor data ensured that we could reliably perform longer-term comparative analysis of heating data across changing conditions, such as seasonal changes, spending less time on data management and more time on research.

5. DISCUSSION

Because of its focus on enforcing the *data integrity principle*, one limitation of the Karibu system is the difficulty of processing or displaying data in real-time. One solution is to provide clients with direct access to the data stream using the messaging system (MQ Cluster in Figure 1) to fan out data flows into additional queues, without affecting the existing flow into the database.

Karibu supports privacy via secure communication from the clients and standard authorization techniques in the messaging and database systems. However, Karibu does not provide any way for users to inspect the raw data collected about them, unless provided by a client UI. In addition, Karibu's centralized storage scheme does not support individual users directly 'owning' their personal data.

Documentation about Karibu, tutorials, and references to the source code can be found at `http://www.karibu.cs.au.dk`.

6. CONCLUSION

We have outlined a set of architectural requirements central to urban IoT data collection, expressed in four architectural design principles: *data integrity*, *modifiability*, *availability*, and *performance*.

We have presented the Karibu architecture and sketched aspects of our open-source reference implementation following the above principles. Karibu provides thin and flexible client side APIs for Android, iOS, and other data producers. It uses a clustered and mirrored message queue system (RabbitMQ) as server side front-end, the Karibu daemons that are responsible for fetching and storing data and allow run-time configuration for new data types, and finally a database system (MongoDB).

Karibu is open-source and we invite the community to use it, and contribute to its further development.

7. ACKNOWLEDGEMENTS

This work has been supported by The Danish Council for Strategic Research as part of the EcoSense project (11-115331) and has been partly funded by the Danish Energy Agency project: Virtual Power Plant for Smartgrid Ready Buildings and Customers (no. 12019). We would like to thank the EcoSense and VPP4SGRBC teams, and our partners including Grundfos, Insero, and Herning Kommune. Thanks also to Peter Urbak for his work on Karibu.

8. REFERENCES

[1] Banker, K. *MongoDB in Action*. Manning Publications Co., 2012.

[2] Blunck, H., Bouvin, N. O., Franke, T., Grønbæk, K., Kjærgaard, M. B., Lukowicz, P., and Wüstenberg, M. On heterogeneity in mobile sensing applications aiming at representative data collection. In *Proc. 1st Intl. Workshop on Pervasive Urban Crowdsensing Architecture and Applications (PUCAA)*, ACM (2013).

[3] Blunck, H., Bouvin, N. O., Mose Entwistle, J., Grønbæk, K., Kjærgaard, M. B., Nielsen, M., Graves Petersen, M., Rasmussen, M. K., and Wüstenberg, M. Computational environmental ethnography: Combining collective sensing and ethnographic inquiries to advance means for reducing environmental footprints. In *Proc. ACM e-Energy '13*, ACM (2013), 87–98.

[4] Brewer, R. S., and Johnson, P. M. WattDepot: An open source software ecosystem for enterprise-scale energy data collection, storage, analysis, and visualization. In *Proc. 1st Intl. Conf. Smart Grid Communications* (2010), 91–95.

[5] Marz, N. Runaway complexity in big data systems... and a plan to stop it. `http://gotocon.com/aarhus-2012/speaker/Nathan+Marz`, 2012. Presentation at GOTO 2012.

[6] Mun, M., Reddy, S., Shilton, K., Yau, N., Burke, J., Estrin, D., Hansen, M., Howard, E., West, R., and Boda, P. PEIR, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proc. MobiSys 2009*, MobiSys '09, ACM (2009), 55–68.

[7] Videla, A., and Williams, J. J. W. *RabbitMQ in Action*. Manning Publications Co., 2012.

[8] Wagner, D. T., Rice, A., and Beresford, A. R. Device analyzer: Large-scale mobile data collection. In *Proc. Sigmetrics Workshop on Big Data Analytics* (2013).