An Architecture for Supporting RAS on Linux-based IoT Gateways

Cu PHAM, Tan LE, Yuto LIM and Yasuo TAN
School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi City, Ishikawa Prefecture, Japan
{cupham, tanld, ylim, ytan}@jaist.ac.jp

***Abstract*—With the rapid advances in IoT technologies, the role of IoT gateways becomes even more important. Therefore, improving the reliability, availability and serviceability (RAS) of IoT gateways is crucial. Nowadays, Linux is widely adopted for core enterprise systems not only because it is a free operating system but also because it offers advantages in regards to operational stability. With many Linux distribution targeting gateways and hubs, a Linux-based IoT gateway is an important part of the future IoT solutions. In this research, an architecture to monitor and mine RAS data from Linux-based IoT gateways is introduced. The proposed framework aims to improve RAS for IoT gateways by handling critical gateway errors as reported by the Linux kernel and provide an error database for further error analysis.**

***Index Terms*—RAS, Reliability, IoT Gateway, Linux**

I. INTRODUCTION

IoT gateways are emerging as the key element of connecting devices to IoT networks and performing critical functions such as device connectivity, protocol translation, data processing, network management and others as described in [1]. Adding IoT gateways with Reliability, Availability, Serviceability (RAS) features would allow the gateways to protect data integrity and help them stay available for longer periods of time.

RAS originated as a hardware-oriented term introduced by IBM to measure mainframe robustness. However, RAS has been extended to other general systems, including software. An overview of RAS is depicted in Fig. 1. Reliability is crucial in every system, event if it comes at the expense of system performance. Slowness can be an acceptable trait of a system, but failure and data loss are unacceptable. Downtime is equally unacceptable, lending to the obvious importance of availability. Finally, serviceability contributes to both of the aforementioned traits, and should help to reduce the ongoing cost of running the system. Many system vendors understand the value of RAS and have introduced RAS solutions in their commercial server offerings. However, there is no vendor-independent RAS solution for IoT gateways, the lack of which can have a significant negative impact on the stability of current and future IoT systems.

In this paper, an architecture that supports RAS features specially for IoT Gateway is introduced. The proposed architecture includes: i) a common layer that is independent from hardware manufacturer to handle critical RAS events/data (errors affect RAS) of IoT Gateways locally, ii) a scalable solution to monitor RAS data from large number of IoT gateways and iii) an infrastructure ready for future big-data integration. A prototype which is based on this proposed architecture was implemented and preliminary results show that this framework was able to locally collect and handle RAS events as well as remotely record RAS data from multiple IoT gateways.

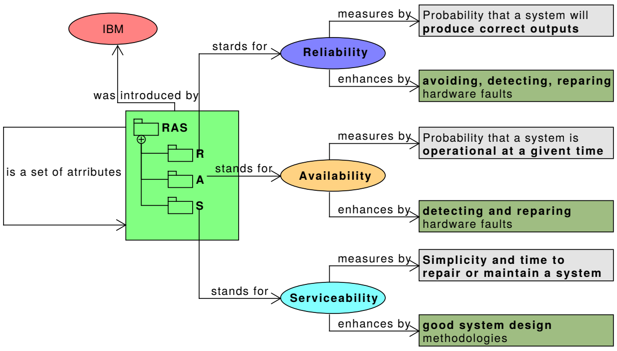
IBM
was introduced by
is a set of atrributes
RAS
R
A
S
stards for
stands for
stands for
Reliability
Availability
Serviceability
measures by
enhances by
Probability that a system will produce correct outputs
avoiding, detecting, reparing hardware faults
Probability that a system is operational at a givent time
detecting and reparing hardware faults
Simplicity and time to repair or maintain a system
good system design methodologies

Fig. 1. RAS Overview

II. BACKGROUND

A. RAS Features

As mentioned above, in practice, there is a lot of commercially available hardware that supports RAS features. All of these features are bound to specific vendor platforms and are constrained by Operating System (OS), CPU type, RAS event types such as CPU temperature and so on. In order to support RAS for IoT gateways in a unified manner, these limitations must be accommodated by implementing a common layer that can be: i) independent from hardware manufacturers, ii) supports operating systems utilized for gateways, iii) handles specific RAS events of gateways.

As shown in Fig. 2, an IoT gateway primarily maintains connections, aggregates data and manages IoT devices. To ensure uninterruptible operation, it is critical to improve the RAS of the IoT gateway. Presently, besides Unix, Linux has a number of features to support RAS. Additionally, there are many Linux distributions with the primary goal of being deployed as gateways, such as OpenWRT, Tizen, etc. Presently, a number of implementations for reporting errors from kernel space

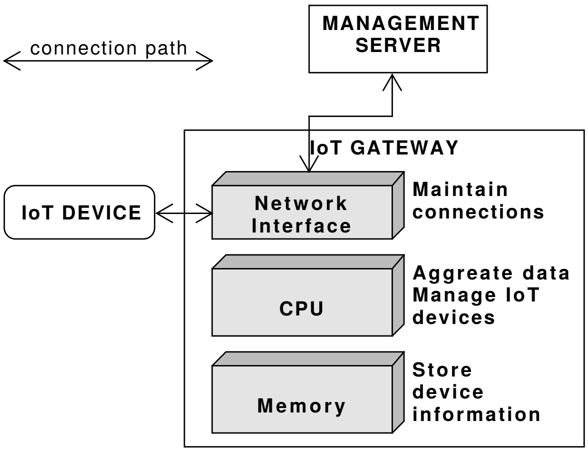

Fig. 2. IoT Gateway Overview

modules to various user space tools already exist for Linux. However, in order to achieve a true interoperable solution, the usage of RAS events reported directly from kernel seems to be the most promising solution. Some kernel space modules that are capable of detecting RAS events from critical features of a gateway are listed as bellow:

- MCE (Machine Check Exception) detects and reports errors related to CPU (processor, cache errors) and depending on the processor it can also provide information regarding memory and bus errors [2].
- EDAC (Error Detection and Correction) is a set of Linux kernel modules for handling hardware-related errors. Its major focus has been ECC memory error handling, while also being able to detect and report PCI bus parity errors [3].
- PCIe AER (Peripheral Component Interconnect Express Advanced Error Reporting) reports AER errors from PCIe hardware (such as network interface card) via a kernel trace event or to console [4].

B. Messaging Technologies

Beside the RAS common layer, a solution to monitor RAS messages from network of gateways must be considered. There are two main approaches in messaging technologies:

- A Broker model that utilizes message brokers to decouple message senders and receivers in order to achieve better scalability and availability. However, it requires excessive amount of network communication and in some cases the broker may become a performance bottleneck.
- A Brokerless model that utilizes direct communication between senders and receivers. This architecture is ideal for applications with a need for low latency and/or high transaction rate as it removes the broker (and one extra network hop) which can be a potential bottleneck. However, in real world enterprise environment with hundreds of interconnected elements, this architecture would quickly become unwieldy and difficult to manage.

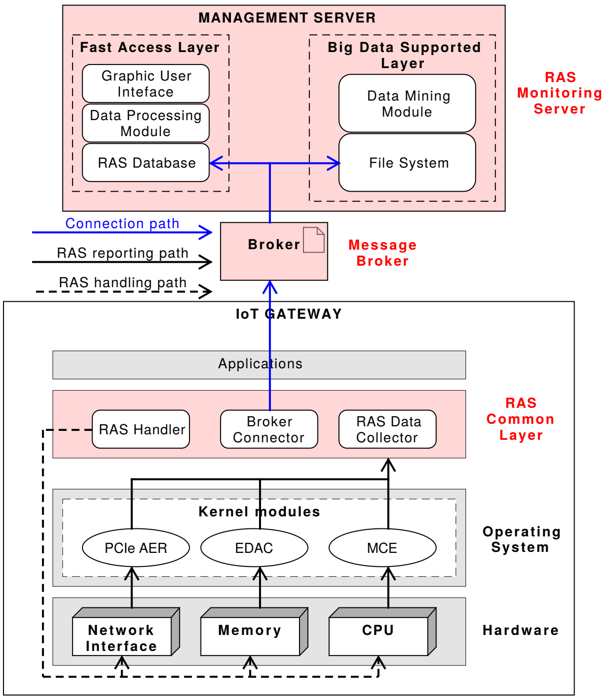

Fig. 3. Proposed Architecture

III. PROPOSED ARCHITECTURE

The objective of this paper is to introduce a RAS common layer to enable RAS features for IoT gateways in a hardware and vendor independent fashion, capable of handling a variety of RAS data and events related to critical functions of a gateway. Furthermore, an architecture to remotely collect and monitor RAS data recorded by the common layer from networks of IoT gateways is proposed. By deploying the proposed architecture, a RAS database can be built for big data analysis purposes (at the time of writing this paper, there is no such RAS database). The proposed architecture includes two main components: i) a **RAS Common Layer**, ii) a **RAS Monitoring Server** and a scalable solution to connect these two main components using **Message Broker** as shown in Fig. 3.

A. RAS Common Layer

The RAS common layer will be installed into gateways for locally collecting, monitoring and handling RAS data reported by kernel. The overview is illustrated in Fig. 4. It consists of the following 3 modules:

1) RAS Data Collector (RDC): This module collects logs generated by kernel modules as input, filters them and classifies hardware events related to critical features of a gateway. This module only outputs RAS data related to operation critical hardware of the IoT gateway

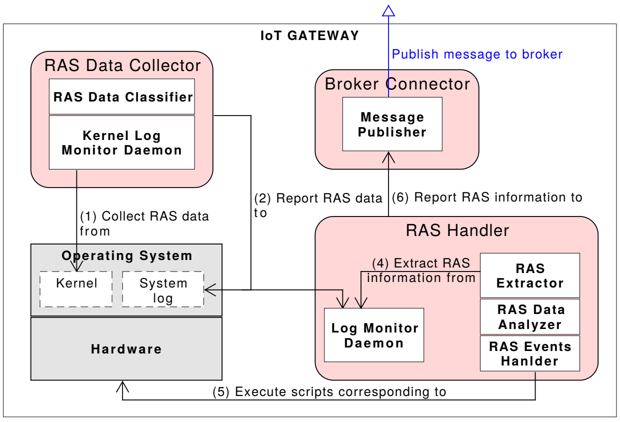

Fig. 4. RAS Common Layer Overview

2) RAS Handler (RH): This module monitors the RAS data reported by RDC module. From this data it extracts RAS information. RH is able to understand the root causes and apply hardware scripts to handle the corresponding errors by hot-swapping defective components or applying fault tolerance patterns.

3) Broker Connector (BC): This module works as a message producer to create and publish messages reported by RH to broker.

B. RAS Monitoring Server

The RAS Monitoring Server (RMS) supports two main modules: i) a fast access layer is responsible for collecting, storing and reporting RAS information to users or administrators via a user-friendly interface; ii) a big data layer is responsible for storing raw data for future applying of big data techniques to analyze RAS data distribution, predict RAS data patterns and so on. Information gained from big data analytic module can be valuable to systems administrators in the future. The detail design of RMS is illustrated in Fig. 5.

C. Message Broker

Currently, the following message brokers can be used to fulfill the purpose of the messaging system:

- **Kafka** is a distributed messaging system providing fast, highly scalable and redundant messaging through a publisher-subscriber model. The distributed design of Kafka has several advantages. First, Kafka allows a large number of permanent or ad-hoc consumers. Second, Kafka is highly available and resilient to node failures and supports automatic recovery. In real world data systems, these characteristics make Kafka an ideal fit for communication and integration between components of large scale data systems.
- **RabbitMQ** implements the Advanced Message Queuing Protocol (AMQP) to act as a message broker. RabbitMQ focuses on the delivery of messages to consumers with complex routing and per-message delivery guarantees. RabbitMQ is more focused on ingesting and persisting massive streams of updates and ensuring that they are delivered to consumers in the correct sequence (for a given partition).
- **Apache ActiveMQ** is a Java based message broker which include a Java Message Service (JMS) client. As with RabbitMQ, the advantage offered by Apache ActiveMQ is the ingesting and persisting of massive streams of data.

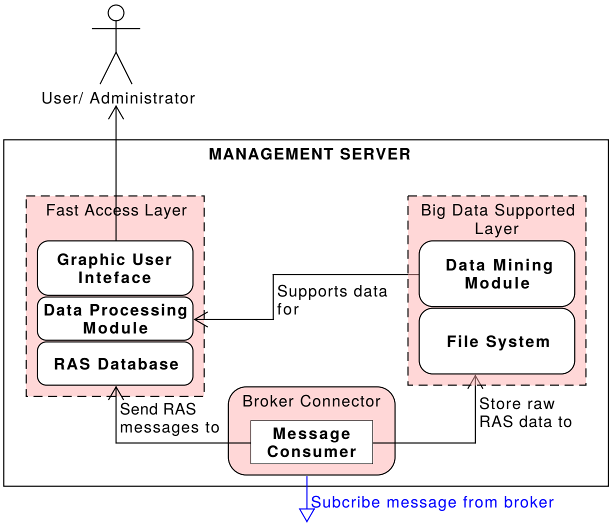

Fig. 5. RAS Monitoring Server Overview

IV. IMPLEMENTATION

To evaluate the proposed architecture, a RAS supported prototype which includes **RAS Common Layer** and **RAS Monitoring Server** by utilizing **Kafka** ecosystem for Linux-based IoT gateway was implemented. The implementation model is illustrated in Fig. 6.

- RAS Local Monitoring Layer is a combination of *ras-daemon* [5] for RAS Data Collector and *fluentd* [6] as RAS Handler and *Kafka* [7] as Server Connector.
- RAS Monitoring Server is a *Nodejs* server [8] which includes *Kafka* module to communicate and exchange RAS data with clients. Data can be stored in a *MongoDB* database [9] in order to be easily accessible via a Graphic User Interface (GUI) or stored long-term in a *Hadoop File System* (HDFS) [10] to be applied data mining techniques in the feature.
- In order to achieve high scalability, the broker model was selected for connecting IoT gateway and RAS Monitoring Server. In this implementation *Kafka* is the selected broker model.

V. SIMULATION AND RESULTS

The deployment diagram of this simulation is visualized in Fig. 7. To verify the prototype system, fake hardware errors must be generated at the IoT gateway, since it is hard to

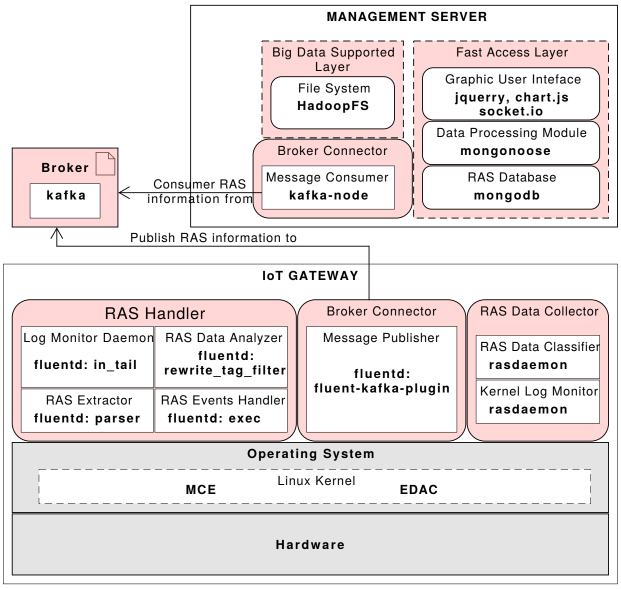

Fig. 6. Prototype Implementation Model

produce and replicate real hardware errors without specialized equipment.

Detail numbers of injected and detected errors are shown in Fig. 8. The result confirmed that the RAS Common Layer was not only able to monitor all RAS data reported by the kernel, but also various type of RAS data. The number of errors recorded by RAS Monitoring Server is reduced when the RAS Local Handler is enabled. This is because the RAS Local Handler disables the faulty hardware and suppresses the reporting of errors already detected by previous RAS events.

An experiment regarding the time necessary for an error to propagate through the system was conducted. The time from when an error was reported by kernel until it was recognized by RDC module was **73 ms**, until it was recognized by RH module was **1653 ms**. It takes a total of **5023 ms** for the BC to sent RAS gateway information to the broker. It takes **27 ms** for the BC at server side to receive the RAS message from the time it was sent by BC at gateway and **35 ms** more to display the information to user via GUI. The average total time for one error reporting cycle is about **5085 ms**.

A. Limitations

Some limitations of this prototype are listed below:

- Since errors related to PCIe-AER are not able to be injected, RAS data related to PCIe-AER can not be simulated.
- Kafka ecosystem is well known for its scalability. However, an experiment involving a large number of clients has not yet been conducted.
- HDFS works efficiently for the analysis and transformation of very large data sets. However, the prototype can not provide such large data sets (lack of experiment machines) in order to verify the effectiveness of HDFS.

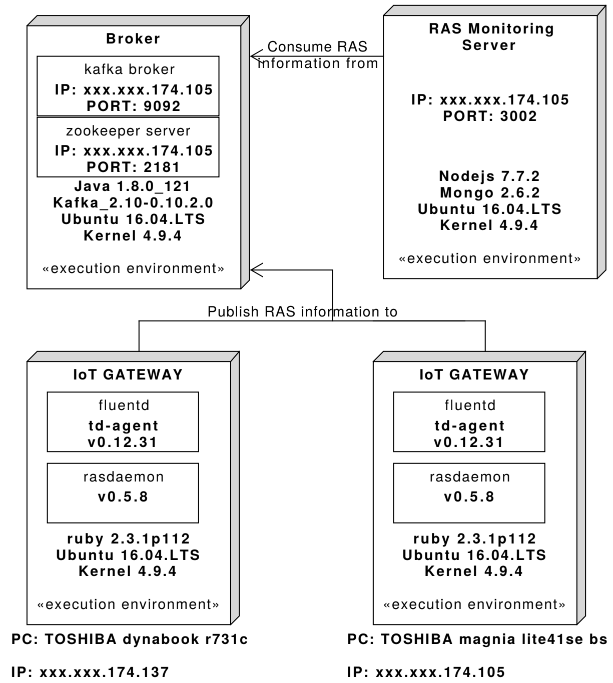

Fig. 7. Deployment Model of the Prototype

VI. CONCLUDING REMARKS

The importance of IoT gateways is increasing as they are responsible for bridging billions of IoT devices to the Internet. Due to this, improving RAS for IoT gateways is of the utmost importance. Currently, there are various solutions to support RAS for commercial server machines. However, there is no hardware independent layer to support RAS feature of the IoT gateway and it still lacks an architecture that is capable of remotely handling RAS data from network of local devices.

This project proposes a complete architecture to monitor and collect RAS data for Linux-based IoT gateways. The proposed architecture provides a RAS Common Layer which is able to monitor various types of RAS data from kernel, support local RAS handler functions to react to hardware errors and is able to report RAS information to the monitoring server. The RAS Monitoring Server is also introduced to allow users/ administrators remotely monitoring RAS features of networks of IoT gateways. Moreover, the proposed architecture supports extensible and scalable infrastructure for supporting RAS data mining in the future by utilizing the *Kafka* and *HadoopFS*

Numbers of Error / Type of Error	Injected Error	Detected by Kernel	Detected by RAS Common Layer	Detected by RAS Monitoring Server	
				Without Local Handler	With Local Handler
MCE	72	72	**72**	**72**	**8**
EDAC	96	96	**96**	**96**	**48**

Fig. 8. Experiment Result: number of injected errors and detected errors

ecosystem.

Data mining techniques which are used to learn and mine RAS data for better error patterns recognition should be extended as future work.

Acknowledgment

This work was partly supported by Toshiba R&D department (IoT Technology Center) through the project to improve reliability for embedded-Linux.

References

[1] *Common requirements and capabilities of a gateway for Internet of things applications*, Telecommunication standardization sector of ITU, 06/2014.

[2] mcelog. [Online]. Available: http://www.mcelog.org/

[3] EDAC Project. [Online]. Available: http://bluesmoke.sourceforge.net/

[4] T. L. N. Yanmin Zhang, “Enable PCI Express Advanced Error Reporting in the Kernel,” *2007 Linux Symposium*, vol. 2, pp. 297–304, 2007.

[5] rasdaemon. [Online]. Available: https://github.com/sujithshankar/rasdaemon

[6] fluentd. [Online]. Available: https://api.ai/

[7] kafka. [Online]. Available: https://kafka.apache.org/

[8] Nodejs. [Online]. Available: https://nodejs.org/

[9] MongoDB. [Online]. Available: https://www.mongodb.com/

[10] Hadoop. [Online]. Available: http://hadoop.apache.org/