Poster: ThingSpire OS: a WebAssembly-based IoT Operating System for Cloud-Edge Integration

Borui Li, Hongchang Fan, Yi Gao and Wei Dong
College of Computer Science, Zhejiang University, and,
Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies, China
{borui.li,fanhc,gaoyi,dongw}@zju.edu.cn

ABSTRACT

We advocate ThingSpire OS, a new IoT operating system based on WebAssembly for cloud-edge integration. By design, WebAssembly is considered as the first-class citizen in ThingSpire OS to achieve coherent execution among IoT device, edge and cloud. Furthermore, ThingSpire OS enables efficient execution of WebAssembly on resource-constrained devices by implementing a WebAssembly runtime based on Ahead-of-Time (AoT) compilation with a small footprint, achieves seamless inter-module communication wherever the modules locate, and leverages several optimizations such as lightweight preemptible invocation for memory isolation and control-flow integrity. We implement a prototype of ThingSpire OS and conduct preliminary evaluations on its inter-module communication performance.

CCS CONCEPTS

• **Computer systems organization → Embedded software**.

KEYWORDS

Operating System, Internet of Things, WebAssembly

ACM Reference Format:
Borui Li, Hongchang Fan, Yi Gao and Wei Dong. 2021. Poster: ThingSpire OS: a WebAssembly-based IoT Operating System for Cloud-Edge Integration. In *The 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '21), June 24–July 2, 2021, Virtual, WI, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3458864.3466910

1 INTRODUCTION

The recent years have witnessed the proliferation of Internet of Things (IoT) technology. Moreover, cloud and edge computing are widely adopted to process the massive data generated by IoT. This computing paradigm leads to a substantial integration of the cloud, edge and IoT applications. Recent advances of computation offloading further improves the application performance by making full use of computing resources. Hence, the cloud-edge-IoT integration is not only the data exchange among devices, but also the portability and interoperability of computing services and modules.

Nevertheless, the evolution of IoT operating system stalls in the recent years and could not keep pace with the growing requirement of IoT applications. For example, using the existing OSes such as Zephyr and RIOT [1] to build an integrated IoT application, developers should follow a separated programming style. This separation indicates that the application logic on different devices should be divided clearly and implemented independently by the developers. Nevertheless, the fixed partition of logic leads to inefficiency and the heterogeneity among cloud, edge and IoT devices makes this separated style error-prone.

We propose *ThingSpire OS*, a novel IoT OS based on WebAssembly tailored to cloud-edge integration. WebAssembly is a bytecode format being introduced by several browser vendors in 2017. It is designed to serve as a common compilation target for high-level languages and execute with near-native performance on a plethora of platforms. This efficiency and portability makes it an ideal underlying technology of ThingSpire OS. However, building such an OS for cloud-edge integration with WebAssembly faces unique challenges. We highlight the challenges and the corresponding solutions as follows:

Challenge 1: How to design an OS that provides coherent execution environment with cloud and edge at system level and runs efficiently on even resource-constrained devices?

Solution 1: By design, we consider WebAssembly as the first-class citizen in ThingSpire OS, which means all the applications is compiled to WebAssembly modules and we implement the WebAssembly runtime inside the OS kernel. Furthermore, we introduce the Ahead-of-Time (AoT) compilation of WebAssembly on resource-constrained IoT devices to facilitate the runtime efficiency. To the best of our knowledge, ThingSpire OS is the first to bring AoT runtime of WebAssembly to IoT devices.

Challenge 2: How to achieve seamless inter-module communication (IMC) considering the modules may locate on the same or different devices?

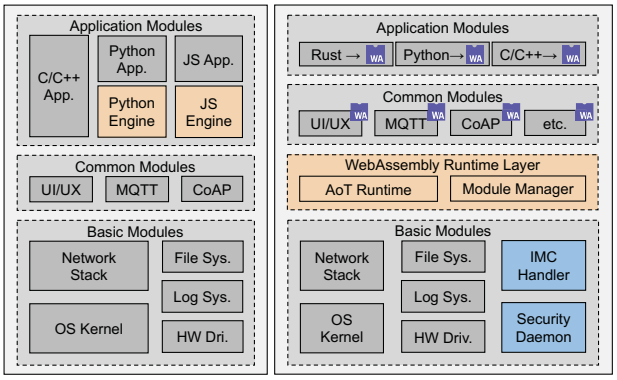
Solution 2: Each module of ThingSpire OS could provide services being invoked via RESTful-style APIs. A service manager handles the invocation routing when the modules migrate between devices and the caller do not need to explicitly specify the actual placement of callee. We further employ zero-copy optimization and network stack bypassing to reduce the inter-module communication overhead.

Challenge 3: Security concern arises when IoT devices tightly integrate with cloud and edge services via network. How to support fault tolerance and memory access safety at system level?

Solution 3: We use a grant based memory sharing mechanism to ensure fine-grained memory isolation. Furthermore, we leverage lightweight preemptible invocation to guarantee the timely return of function calls, and use contract-based interfaces for fault isolation between modules.

MobiSys '21, June 24–July 2, 2021, Virtual, WI, USA

ACM ISBN 978-1-4503-8443-8/21/06.
https://doi.org/10.1145/3458864.3466910



(a) Traditional IoT OS (b) ThingSpire OS

Figure 1: System architecture comparison of IoT OSes.

This paper presents the initial design, implemented prototype and preliminary evaluation of the effectiveness of ThingSpire OS.

2 DESIGN OF THINGSPIRE OS

Figure 1 illustrates the architecture comparison of traditional IoT OS (we take Zephyr as a representative) and ThingSpire OS. As an OS tailored to the IoT application with cloud-edge integration, ThingSpire OS compiles the applications and common modules to WebAssembly and leverages a WebAssembly Runtime Layer to support their execution, which is different from supporting cross-platform languages in the application layer of traditional IoT OSes (Figure 1(a)). By taking advantage the cross-platform nature of WebAssembly, these modules could be seamlessly migrated to the cloud and edge servers. Furthermore, WebAssembly provides basic isolation between modules, which is a good starting point to protect the modules in ThingSpire OS from being compromised.

There are four building blocks in ThingSpire OS: AoT Runtime, Module Manager, Inter-Module Communication (IMC) Handler and Security Daemon.

The AoT Runtime is responsible to provide execution environment of WebAssembly. There exist several WebAssembly interpreters such as `wasm3` [3]. However, these interpreters incur over 10× runtime overhead, which lengthens the active time of IoT devices and shortens their battery lifetime. The AoT Runtime works as follows: (1) Each time a module is loaded, the AoT runtime compiles the WebAssembly bytecode to native assemblies. (2) After the AoT compilation, our runtime does several optimizations to eliminate unnecessary native instructions to further speedup the execution. For example, we remove the consecutive push/pop pairs such as `PUSH(R1)` and `POP(R1)` generated by the translation of two successive WebAssembly bytecodes because this instruction pair is redundant in terms of functionality.

The IMC Handler and the Module Manager work jointly for the efficient IMC in ThingSpire OS. The Module Manager is in charge of the migration of modules among IoT, edge and cloud. Once the placement of a module changes, the Module Manager registers the location to IMC Handler. The IMC Handler processes all the IMC invocations and dynamically routes the IMCs to the corresponding destination. For the local IMCs, our handler generates a non-network bypass to avoid the overhead to go through

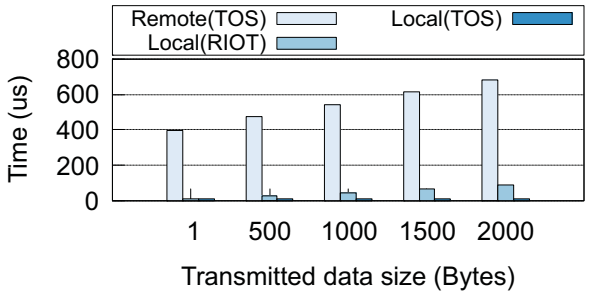


Figure 2: Performance comparison of remote-IMC and local-IMC between ThingSpire OS (TOS) and RIOT OS.

network stack. IMC Handler leverages zero-copy optimization for both remote and local IMCs to further reduce the communication overhead.

The Security Daemon contains several software-based protections. Based on the basic inter-module isolation provided by WebAssembly, ThingSpire OS focuses on the control-flow integrity (CFI) and memory isolation on IoT devices without memory management unit (MMU). To keep the CFI of function invocations across WebAssembly Runtime Layer (mostly system calls and peripheral interactions), ThingSpire OS employs a lightweight preemptive execution technique to guarantee the timely return of function calls, which guarantees the invocations are not perpetually trapped in the buggy or malicious peripheral drivers. For the memory isolation of modules, based on the basic isolation provided by WebAssembly, ThingSpire OS further provides the memory sharing mechanism with grants, which provides the fine-grained memory isolation.

3 PRELIMINARY EVALUATION

We implement an initial version of ThingSpire OS based on the architecture presented in Figure 1 on ESP32 [2], a widely used IoT node integrated with WiFi and Bluetooth connectivity. The initial prototype is developed using C and C++. We also conduct preliminary evaluations on the performance of inter-module communication. We can see from the results in Figure 2 that compared to existing OS (RIOT), ThingSpire OS achieves more efficient IMC by leveraging zero-copy optimization.

4 CONCLUSION

This paper presents a novel IoT operating system targets on cloud-edge integration named ThingSpire OS. ThingSpire OS employes AoT WebAssembly runtime to improve the execution efficiency, uses IMC Handler to manage the inter-module communication, and leverages several security mechanism to ensure the safety of the entire system.

REFERENCES

[1] Emmanuel Baccelli, Oliver Hahm, Mesut Günes, Matthias Wählisch, and Thomas C Schmidt. 2013. RIOT OS: Towards an OS for the Internet of Things. In *Proc. of IEEE INFOCOM WKSHPS*.
[2] ESPRESSIF. 2020. ESP32. Website. https://www.espressif.com/en/products/socs/esp32.
[3] Wasm3 Labs. 2020. wasm3. Website. https://github.com/wasm3/wasm3.