# Shyper: An embedded hypervisor applying hierarchical resource isolation strategies for mixed-criticality systems

Yicong Shen 1st, Lei Wang 2nd, Yuanzhi Liang 3rd, Siran Li 4th, Bo Jiang 5th
School of Computer Science and Engineering
Beihang University
Beijing 100191, PR China
{shenyicong1023, wanglei, liangyz, ohmrlsr, jiangbo}@buaa.edu.cn

***Abstract*—With the development of the IoT, modern embedded systems are evolving to general-purpose and mixed-criticality systems, where virtualization has become the key to guarantee the isolation between tasks with different criticality. Traditional server-based hypervisors (KVM and Xen) are difficult to use in embedded scenarios due to performance and security reasons. As a result, several new hypervisors (Jailhouse and Bao) have been proposed in recent years, which effectively solve the problems above through static partitioning. However, this inflexible resource isolation strategy assumes no resource sharing across guests, which greatly reduces the resource utilization and VM scalability. This prevents themselves from simultaneously fulfilling the differentiated demands from VMs conducting different tasks. This paper proposes an efficient and real-time embedded hypervisor "Shyper", aiming at providing differentiated services for VMs with different criticality. To achieve that, Shyper supports fine-grained hierarchical resource isolation strategies and introduces several novel "VM-Exit-less" real-time virtualization techniques, which grants users the flexibility to strike a trade-off between VM's resource utilization and real-time performance. In this paper, we also compare Shyper with other mainstream hypervisors (KVM, Jailhouse, etc.) to evaluate its feasibility and effectiveness.**

***Index Terms*—embedded system; mixed-criticality; virtualization; resource utilization; real-time**

## I. INTRODUCTION

With the continuous development of the Internet of Things and increasingly fierce competition in market, it has become a critical problem [1] to make embedded products integrate a variety of functions with different reliability, real-time performance and verification requirements while under the constraints of power consumption, volume, weight and cost. Traditional Solutions, such as AUTOSAR can only provide an open and standardized software architecture that allows manufacturers to freely integrate various software components, but they cannot guarantee the isolation between their interfaces [2]. Modern embedded systems are evolving to general-purpose and mixed-criticality systems. It is necessary to classify tasks according to their requirements and place them into specified domains, ensuring the isolation between different functions. Isolation and non-intervention between these domains have become a key requirement.

In recent years, the popularity of multi-core processor architecture enables embedded software to have enough computing resources to simultaneously support parallel computing for multiple complex tasks. However, the traditional hardware design of multi-core processors with real-time operating systems is incapable of guaranteeing the temporal and spatial isolation of multi-tasks, leading to a big waste of resources or potential safety hazard. With the popularity of multi-core processor architecture and the support of reduced instruction set computing (RISC) for hardware virtualization, it has become possible to use virtualization to establish multiple isolated partitions accommodating various functions in embedded systems such as automation and UAV [3]. Virtualization introduces a software layer called hypervisor or virtual machine monitor (VMM) between the hardware layer and the application layer to provide spatial and temporal isolation. Through the hypervisor, the VMs can run independently and use the hardware resources abstracted by the hypervisor without modification. Virtualization allows: 1) dynamic addition and removal of partitions without affecting other partitions, 2) time-sharing and multiplexing of underlying device resources among partitions, and 3) preventing malicious attacks and data leakage between partitions [4].

Although promising, implementing embedded virtualization is still a challenging task. Unlike servers or personal computers, embedded hardware tends to have tight constraints in cost, weight, volume, power consumption, and real-time, etc. Besides that, embedded hypervisors have to be adapted to many legacy embedded RTOSes, such as Zephyr, QNX, Nucleus, VxWorks, etc. However, traditional server-based hypervisors, like KVM and Xen could not be easily restructured to meet the requirements mentioned above. In recent years, these challenges motivate the appearance of various embedded hypervisors, most of which apply the strategy of static resource partition and effectively reduce the real-time performance overhead and the Trusted Code Base (TCB), making them better choices in the embedded field. However, this strategy also reduces the resource utilization and VM scalability, thus hindering them from simultaneously fulfilling the differentiated demands from VMs conducting different tasks.

To resolve the problems mentioned above, we propose "Shyper", a new type I real-time embedded hypervisor supporting hierarchical resource isolation strategies. The contributions of our work are as follows:

- To provide differentiated services for VMs with various criticality and requirements, Shyper proposes hierarchical isolation assignment strategies through manually classifying VM according to scenarios and adopting fine-grained spatial and temporal resource isolation mechanisms within each classification.
- To further guarantee the real-time performance, Shyper proposes two novel "VM-Exit-less" real-time virtualization techniques: Task Remote Core Assignment (TRCA) and GIC Partial Pass-through (GPPT). These techniques grant users the flexibility to make a trade-off between VM's resource utilization and real-time performance.
- Finally, we have systematically evaluated Shyper against other mainstream embedded hypervisors (KVM, Jailhouse, etc.) in the industry to evaluate its the feasibility and effectiveness.

## II. Related Work

### A. Traditional Server-based Virtualization

The main design goal of traditional server-based hypervisors (KVM, Xen) is to provide efficient, fair and scalable virtual machine services. As a result, the mechanisms of these designs are inadequate to be directly applied in the mixed-criticality scenario. For example, the traditional VM schedulers do not take VMs' different criticality into consideration and fail to provide sufficient differentiation. In recent years, many studies have tried to introduce new VM scheduling algorithms so that KVM and Xen can meet the basic real-time constraints, which have achieved some success [5]. However, these scheduling algorithms cannot fully eliminate the temporal uncertainty introduced by VM scheduling, and by the way there are still other mechanisms or strategies that deeply affected the real-time performance of the VMs running above.

In addition to the scheduling, traditional server-based hypervisors still have potential factors such as memory allocation and device models (DM) that can destructive break the real-time constraints. For example, the memory management API of Xen is very powerful but its implementation is extremely complex. When running applications with heavy memory load, frequent VM Exits caused by page faults will introduce a lot of time uncertainty.

Furthermore, it is possible that the task with high priority in a VM can be blocked by the task with low priority in another VM under certain circumstances, which is called priority inversion. A common case in Xen is that when there is a high load in Dom0 and the processes related to DM cannot get enough CPU clock cycles to handle the I/O requests, the tasks requiring I/O services in DomU are blocked in a sense [6].

Finally, traditional server-based hypervisors often have a huge TCB, and some mechanisms are too complicated for embedded systems to optimize, and it is difficult to pass the common security certification in the embedded field.

### B. Embedded Virtualization Base on Static Partition Strategy

To get rid of the burden of history, several new virtualization architecture design ( Jailhouse, Bao, Acrn) have been proposed in the recent years. These designs are based on the concept of static partitioning, which means statically partitioning all platform resources and assign each one exclusively to a single VM instance [7]. Through this strategy, these hypervisors succeed in greatly reducing the overhead introducing by VM scheduling, memory allocation, device model and other mechanisms, thus providing much stronger real-time performance than the traditional designs [8], [9]. In addition, this strategy greatly simplifies the mechanisms and reduces the TCB of the hypervisors, making them relatively lightweight and easy to verify.

However, the static partitioning strategy assumes no hardware needs to be shared across guests and thus results in low resource utilization. For example, both Jailhouse and Bao apply the pass-through only IO configuration, making it impossible to multiplex a single block device between multiple VMs. Besides that, as each vcpu of a VM need to be exclusively assigned with a pcpu, the CPU utilization cannot be easily improved through overcommitment. To make matters worse, the low resource utilization and the lack of VM scalability can eventually lead to the impossibility of finding an available VM configuration that can fulfill every VM's demands due to the tight constraints of volume, weight and cost in embedded systems.

In some cases, the design principle of static partition is forced to be broken since some of the essential hardware have to be multiplexing between different VMs. Take Jailhouse and Bao as examples, the implementations of both two hypervisors fail to eliminate the trap-and-emulated overhead and interrupt-inject latency when the target platform is equipped with GICv2/v3.

## III. The Design of Shyper

### A. System Structure

Shyper is designed to be an efficient, scalable and real-time embedded hypervisor and its design goals are as follows:

- Shyper should guarantee strong isolation between VMs;
- Shyper should be lightweight in both TCB and memory footprint;
- Shyper should provide differentiated services for VMs through hierarchical isolation strategies;
- Shyper should guarantee the real-time performance of critical VMs;

In order to achieve the goals above, as is shown in Figure 1, Shyper classifies VMs into four types with different scenarios, criticality and real-time constraints:

- Management VM (MVM), which is similar to Xen's Dom0 design, is used to provide an external control interface for administrators and native driver services for other types of VMs; it is worth noticing that MVM is not supposed to be indispensable in practice;
- Guest VM (GVM) is used to run a general operating system, such as Linux, Android, Windows, etc., with non-critical tasks like scientific calculation and so on;
- Soft Real-time VM (SRTVM) is used to hold a real-time operating system (RTOS), such as PREEMPT_RT Linux, with soft real-time tasks running on it.

- Hard Real-time VM (HRTVM) provides the strongest real-time performance which is designed to run hard real-time tasks, such as RTOS or bare-metal applications.

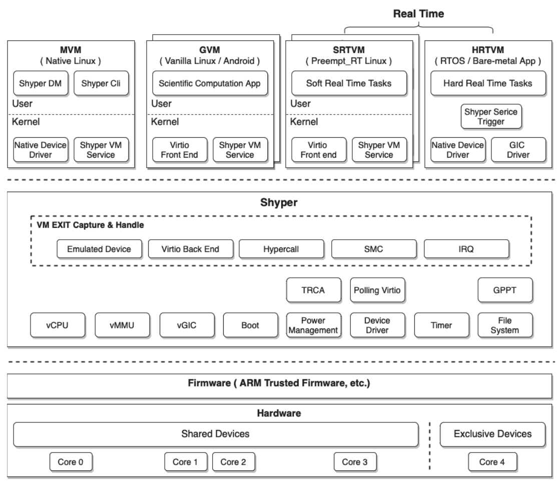

Fig. 1. System Structure of Shyper, which is designed to be a Type I real-time hypervisor running at EL2 privilege level, capturing and handling all kinds of exceptions like data abort, hvc, smc, irq, etc..

To guarantee the strong isolation between VMs and provide differentiated services, Shyper applies hierarchical resource isolation strategies through designing fine-grained spatial and temporal isolation mechanisms for each type of VM, which are elaborated later.

To reduce the code size of Shyper as much as possible, Shyper implements a VM management interface within MVM, allowing administrators to configure, boot and destroy VMs. In addition, Shyper also implements mediated pass-through device model in MVM to reuse the native device driver code in a large amount, eliminating the trouble of driver transplantation. Besides that, Shyper also provides NAT network services for GVM and SRTVM by configuring Virtio network in all these VMs and forwarding Ethernet frames between them, making MVM play as a gateway and multiplexing its network protocol stack.

To realize high resource utilization and rich scalability, Shyper has implemented a wealth of device models (DM), such as Virtio, full emulation, pass-through and mediated pass-through devices, providing multiple types of devices including serial, storage, network and so on. Although there are many advantages to implement mediated DM in MVM, Shyper also implements it partially in its kernel to meet the possible low-latency and reliable I/O service requirements for critical VMs.

In order to support real-time virtualization, Shyper introduces two novel real-time virtualization techniques: 1) Task Remote Core Assignment (TRCA) mechanism which can simultaneously ensure the resource utilization and real-time performance for SRTVM; 2) GIC Partial Pass-through (GPPT) which can almost eliminate both the overhead of trap-and-emulation for vGIC and the latency caused by interrupt injection. These two techniques will be introduced in detail later.

## *B. Hierarchical Resource Isolation Strategies*

Spatial and temporal isolation are important features to ensure that VM access is limited to resources allocated to itself (memory, registers, etc.). In embedded virtualization, it is necessary to ensure that IoT devices can safely collect and process user data, which cannot be obtained or destroyed by other malicious VMs. Shyper mainly guarantees the isolation through CPU and memory isolation.

Shyper implements different vcpu scheduling algorithms in CPU partitions corresponding to different VM classifications. Firstly, considering that MVM needs to process disk and network requests from other VMs, and the pcpu where MVM is located is likely to be used to support special tasks and frequent VM scheduling will a cause rapid decline in overall I/O performance, so a fixed allocation strategy is chosen in MVM partition. Secondly, the main consideration of GVM partition is the fairness of the VM scheduling strategy, where a group round-robin VM scheduling strategy is selected. Then, SRTVM needs to ensure certain real-time constraints, and can use predictable scheduling algorithms that have been widely used in KVM and RT-Xen. Finally, HRTVM needs to ensure absolute real-time performance, and chooses to adopt a fixed core allocation strategy consistent with MVM, except that HRTVM tends to be a single-core system.

In order to ensure data security between VMs, Shyper uses the memory isolation provided by the two-stage address translation mechanism in ARMv8 to ensure that the memory data of each VM cannot be obtained or modified by other VMs. To reduce the TCB and minimize the time uncertainty introduced by dynamic memory allocation, Shyper adopts the following three strategies: 1) Direct address space mapping, which means that, including Shyper, MVM and all RTVMs, the intermediate physical address (IPA) is completely consistent with the actual physical address (PA) or maintains a fixed offset; For GVM, dynamic memory allocation is still applied to improve the overall memory utilization; 2) All page tables for stage 2

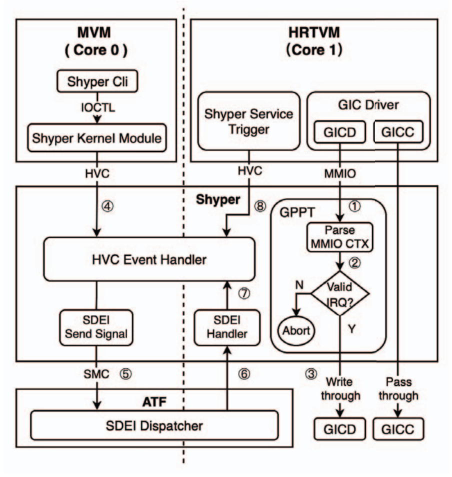

Fig. 2. GPPT Implementation

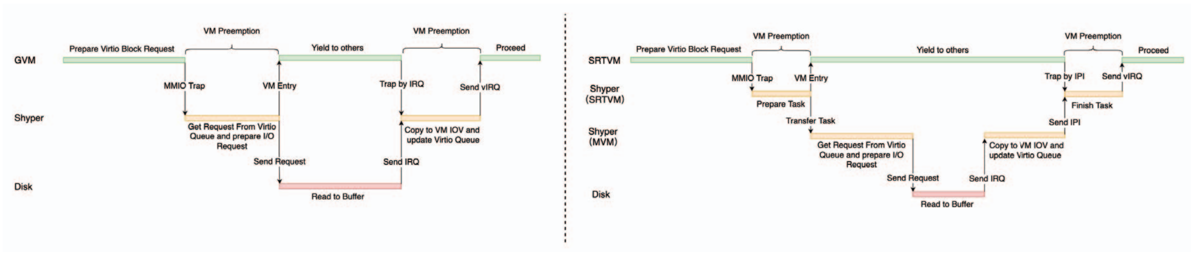

Fig. 3. An example of TRCA usage in Virtio Disk Read

address translation of memory and devices are pre-establish before each VM is booted, except for the emulated devices. It avoids the huge overhead introduced by the frequent VM Exit and the complex dynamic memory allocation mechanism in Xen.

### C. Real-Time Virtualization Mechanisms

*1) GIC Partial Pass-through:* GIC partial pass-through (GPPT) is a technology that can be used to solve interrupt delays and frequent VM Exits caused by interrupt virtualization. The core idea of GPPT is to reserve part of the interrupts of a complete interrupt controller, which will not be intercepted and simulated by the hypervisor, but is directly passed to the upper VM. Hypervisor only needs to ensure that the VM using GPPT is isolated from other VMs's interrupt when a configuration interrupt properties through the GIC driver.

Figure 2 shows the realize of GPPT. Similar to traditional hardware-assisted vGIC virtualization, it needs to trap all MMIO access to GICD from the VM but the difference is Shyper will pass its own interrupt configuration (defined by the configuration list) directly to GICD instead of implementing vGIC, as shown by step 1-3. Another difference is that GICC instead of GICV is passed through to the VM equipped with GPPT, and the pcpu assigned to it needs to configure HCR_EL2.IMO and HCR_EL2.AMO to make all external interrupts directly injected into EL1 privilege level instead of EL2.

To ensure the isolation of interrupt resources, Shyper only helps the upper VM to pass its own interrupt configuration which is defined by the VM configuration.

Another problem that GPPT introduces is that the pcpu configured with GPPT can no longer intercept any interrupts through the hypervisor, and can not inject virtual interrupts through vGIC, which directly leads to the loss of Shyper's control for the pcpu and the VM running on it. Shyper uses two different methods to solve this problem: 1) If the target platform can provide ARM Trusted Firmware (ATF), Shyper can regain control of the target core through the Software Delegated Exception Interface (SDEI) asynchronous event notification mechanism, as shown by step 4-7. 2) If ATF is not available, it is necessary to use a user mode polling mechanism based on shared memory between MVM and the target to ensure that VM management and the inter-vm communication still work, as shown by step 8.

*2) Task Remote Core Assignment:* Task Remote Core Assignment (TRCA) is a new technique that can be used to solve the time uncertainty caused by the various non-preemptible hypervisor blocking events in Section 2.2. When VM is running, the hypervisor needs to handle a variety of synchronous and asynchronous requests from the upper VM, including device simulation, hypercall, SMC and so on. When switching from VM to hypervisor layer, all tasks in the VM sending the request are completely frozen, which might cause some critical tasks to miss their deadlines. In order to reduce the impact of these events on VM's real-time performance, there is a novel method to consider: transforming synchronous requests to asynchronous requests and send these requests to a specific pcpu to process.

In this article, we introduce the design of "Task Remote Core Allocation" in Shyper based on the third optimization idea. Unlike the Dom0 design in Xen, which may have high-priority VMs blocked by low-priority tasks in Dom0, Shyper does not put the code logic into a specific VM to complete, but by preempting certain pcpu where MMV or a low-priority VM is located to complete the task. TRCA consists of four parts: 1) task generation module, 2) task scheduler, 3) task receiving queue, 4) task execution unit.

Figure 3 shows the respective Virtio Block request processing flow of GVM and SRTVM. For GVM, there is a large amount of executions taking place in hypervisor, including extracting requests from the Virtio queue, I/O vector merging, data copying, and driver interaction, etc.. These executions blocked the normal operation of GVM for a large proportion of overall running time. When it comes to SRTVM which applies TRCA mechanism, most of the time-consuming execution is forwarded to the proxy pcpu for completion and only left the task preparation and completion, which greatly reduces the amount of time remaining in the hypervisor layer.

TABLE I
INTERRUPT LATENCY.
UNIT: NS, MEASURED AT THE SAME CPU FREQUENCY.

| | Min | Avg | Max | Std | Normed Avg |
|---|---|---|---|---|---|
| Native | 96 | 96 | 128 | 7.54 | 1 |
| GPPT | 512 | 542 | 690 | 35.28 | 5.33 |
| vGIC | 13440 | 13488 | 18240 | 535.61 | 140.05 |

TABLE II
VM EXIT OVERALL OVERHEAD. UNIT: US, INJ - EXTERNAL INTERRUPT INJECTION / IPI - INTER-CORE INTERRUPT / MT - MAINTENANCE INTERRUPT / IBLK - DISK INTERRUPT / INTC - INTERRUPT CONTROLLER / VBLK - VIRTIO DISK / HVC - HYPERCALL
TEST DURATION: 619768 US

| Event | MVM | | GVM | | SRTVM | | HRTVM | |
|---|---|---|---|---|---|---|---|---|
| | Count | Total | Count | Total | Count | Total | Count | Total |
| INJ | 1823341 | 2707523 | 9765 | 12424 | 9000 | 11364 | 0 | 0 |
| IPI | 2427158 | 9272874 | 3331350 | 5913098 | 380568 | 480966 | 0 | 0 |
| MT | 1665799 | 937657 | 1665484 | 913623 | 380574 | 227582 | 0 | 0 |
| IBLK | 381063 | 1686958 | 0 | 0 | 0 | 0 | 0 | 0 |
| INTC | 40 | 63 | 0 | 0 | 1 | 1 | 0 | 0 |
| VBLK | 0 | 0 | 4996510 | 4332174 | 1141851 | 611485 | 0 | 0 |
| HVC | 1665738 | 1712779 | 0 | 0 | 0 | 0 | 0 | 0 |
| Total | | 16318 | | 11171 | | 1331 | | 0 |
| Percentage | | 2.63% | | 1.80% | | 0.21% | | 0.00% |

TABLE III
TEST RESULTS OF CYCLICTEST, UNIT: MS.

| Target | Min | Avg | Max | Std-dev |
|---|---|---|---|---|
| Native-L4T | 5 | 11 | 35 | 2.39 |
| Shyper-MVM | 4 | 11 | 42 | 2.20 |
| KVM-Guest | 8 | 15 | 1022 | 5.94 |
| Shyper-GVM | 4 | 10 | 554 | 12.34 |
| Jailhouse-NoneRootCell | 5 | 9 | 26 | 2.45 |
| Shyper-SRTVM | 4 | 9 | 20 | 1.60 |
| Shyper-HRTVM | 3 | 6 | 15 | 1.26 |

## IV. EXPERIMENTAL RESULTS

### A. Experiment Setup

To make a comparison, we selected KVM and Jailhouse as the experimental objects, which are two mainstream hypervisors that can be successfully launched on ARMv8-based Nvidia Jetson TX2 development board. In terms of virtualization overhead, we design experiments to investigate the interrupt latency and VM Exit performance overhead. As for real-time performance, Cyclictest has become the only choice. In addition, when the system needs to be tested under a certain load, Stress is used to provide the corresponding load.

### B. Virtualization Overhead

For any hypervisor, the interrupt latency and the cost of VM EXIT directly determine the computing and real-time performance of the VM. In this paper, the two mechanisms GPPT and TRCA are introduced to minimize these two overheads.

Firstly, to accurately measure the interrupt latency, we have transplanted and revised FreeRTOS so that it can run directly on TX2, run as HRTVM with GPPT and GVM with vGIC at the same time. Then, We measure the interrupt delay of all three through Generic Timer, shown in Table I.

Comparing with the native FreeRTOS running on TX2, the GPPT and vGIC respectively show about 5 times and 140 times higher latency. From the results, although GPPT still has a certain gap compared with the native, it successfully reduces the interrupt latency to 1/28 of vGIC by optimizing the extra overhead of interrupt injection and vcpu context switching in the case of vGIC.

After the measurement of the external interrupt latency, we designed an experiment to further quantitatively analyze the additional performance overhead caused by VM EXIT. Four different types of VMs were launched and imposed CPU, memory, and disk loads at the same time. During the test, Shyper was responsible for recording all types of VM EXIT events, including the frequency and the time spent in Shyper.

Table II shows the frequency and overall overhead introduced by VM EXIT during the whole test. Due to the existence of TRCA, different type of VMs has completely different behaviors: 1) MVM needs to provide GVM with mediated disk services and its core needs to handle remote tasks (mainly disk I/O tasks) for SRTVM, the VM EXIT overhead of which mainly comes from interrupt injection, inter-processor interrupts, and disks interrupt and HVC. In general, the pcpu where MVM is located has 2.63% of the overall time is staying in hypervisor layer; 2) The main VM EXIT overhead of GVM comes from the mediated disk devices, which is mainly reflected in the inter-processor interrupt, maintenance interrupt and the emulation of Virtio disks, occupying 1.8% of the overall time; 3) The main VM EXIT overhead of SRTVM is similar to GVM, which also comes from the inter-processor interrupt, maintenance interruption and Virtio disk emulation, but due to the TRCA mechanism, its inter-processor interrupt and Virtio disk emulation overhead is much smaller than GVM. As a result, during the entire test the core where SRTVM is located only has 0.21% time staying in Shyper, only the 8% of MVM case and the 11% of GVM the respectively; 4) Due to the usage of GPPT and all pass-through devices method, HRTVM did not generate any VM EXIT during the whole test.

Due to TRCA, almost all the time-consuming tasks of SRTVM are reallocated to the pcpu where the MVM is located, only left the overhead of the request initiation and the result confirmation. From the test result, SRTVM limits the average VM Exit time within 1us, and the maximum value to less than 5us through completely eliminating the originally inevitable overhead.

### C. Real-time Performance

For the real-time performance evaluation, we use Cyclictest to compare Shyper with L4T, KVM, Jailhouse In order to eliminate the influence from OS itself, we have tried our best to use the similar version of PREEMPT_RT Linux to replace the kernel originally used by each VM.

In order to prevent effect introduced by the multi-core architecture, every Linux's bootarg is set as "isolcpus = 1

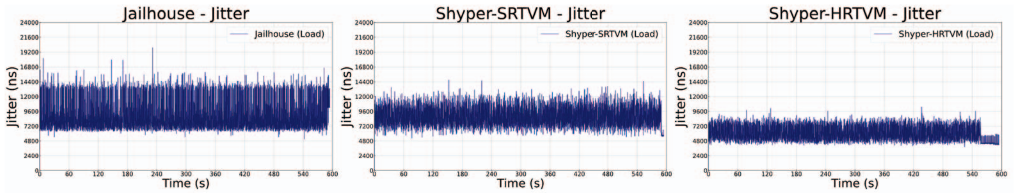

Fig. 4. Cyclictest Jitter.

irqaffinity = 0 idle = poll". In addition, the process of QEMU in KVM host is fixed to core 0 through taskset. Since it is impossible to unify the kernel version of all VMs, we still group the tests according to the kernel version and VM type. All tests were conducted for 600s. During the test, CPU, memory and I/O loads were applied through Stress to emulate real working conditions.

As shown in Table III, the real-time performance of L4T and Shyper-MVM is similar, except that the worst latency of Shyper-MVM is slightly higher than that of the native. However, it is worth noticing that when Shyper-MVM is conducting mediated devices workloads, or undertaking the remote tasks of SRTVM or HRTVM, the real-time performance will quickly degrade to the level of GVM. The KVM is the worst of all three, which may result from the host Linux scheduling for QEMU device model. Shyper-GVM's worst delay will be better than KVM but still far worse than Native because of the implementation of vGIC and Virtio devices.

Figure 4 shows the jitter varying with time. Because Jailhouse adopts the "VM-Exit-less" design idea, both of its worst case latency and std-dev of delay perform well, while the limitation of the GICv2 make it impossible to completely block the VM EXIT related to interrupts from external devices. Shyper-SRTVM's TRCA can ensure the VM EXIT caused by emulated devices to be reduced as much as possible. Under the pressure of I/O workload, it still performs better than Jailhouse in every testing metric. Both Jailhouse and Shyper-SRTVM still use MMIO-trap-based vGIC emulationO. However, HRTVM uses the GPPT mechanism to further reduce the interrupt latency, which has the lowest delay and jitter among all three tests.

## V. Future Work

First of all, the embedded software tends to have higher requirements for safety and security. One of our goals is to make Shyper pass the EAL5+ international information security certification standard. In addition, the Trusted Execution Environment (TEE) is widely used to improve the security of embedded systems such as Android. We have planned to use ARMv8's TrustZone technology to introduce the TEE in Shyper, to provide security-related services. Finally, we plan to bring Shyper to more hardware platforms and architectures (x86, RISC-V, etc.) and because Shyper does not rely on Linux as MVM, it can support more different types of OS in the future.

## VI. Conclusion

This article introduces Shyper, an embedded hypervisor designed for mixed-criticality systems. Through hierarchical resource isolation strategies and effective real-time optimization mechanisms, Shyper strikes a balance between resource utilization and real-time performance for embedded computing scenarios. The result of our experimental evaluation confirms the resource utilization and real-time performance of Shyper as a generic embedded virtualization solution.

## Acknowledgment

This work was partially supported by National Natural Science Foundation of China (No. 62077002)

## References

[1] G. Weiss, P. Schleiss, and C. Drabek, "Towards flexible and dependable e/e-architectures for future vehicles," in *4th International Workshop on Critical Automotive Applications: Robustness & Safety (CARS 2016).*

[2] D. Reinhardt, D. Kaule, and M. Kucera, "Achieving a scalable e/e-architecture using AUTOSAR and virtualization," vol. 6, no. 2, pp. 489–497, number: 2013-01-1399. [Online]. Available: https://www.sae.org/publications/technical-papers/content/2013-01-1399/

[3] M. Strobl, M. Kucera, A. Foeldi, T. Waas, N. Balbierer, and C. Hilbert, "Towards automotive virtualization," in *2013 International Conference on Applied Electronics*, pp. 1–6, ISSN: 1803-7232.

[4] G. Heiser, "The role of virtualization in embedded systems," in *Proceedings of the 1st workshop on Isolation and integration in embedded systems*, ser. IIES '08. Association for Computing Machinery, pp. 11–16. [Online]. Available: https://doi.org/10.1145/1435458.1435461

[5] J. Yang, H. Kim, S. Park, C. Hong, and I. Shin, "Implementation of compositional scheduling framework on virtualization," vol. 8, no. 1, pp. 30–37. [Online]. Available: https://doi.org/10.1145/1967021.1967025

[6] L. Abeni and D. Faggioli, "Using xen and KVM as real-time hypervisors," vol. 106, p. 101709. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762120300035

[7] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems," in *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*, ser. OpenAccess Series in Informatics (OASIcs), M. Bertogna and F. Terraneo, Eds., vol. 77. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 3:1–3:14, ISSN: 2190-6807. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/11779

[8] M. Masmano, I. Ripoll, A. Crespo, and J. Metge, "Xtratum: a hypervisor for safety critical embedded systems," in *11th Real-Time Linux Workshop*. Citeseer, 2009, pp. 263–272.

[9] Z. Jiang, N. C. Audsley, and P. Dong, "Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* IEEE, 2018, pp. 75–84.