RJMM: Real-time Enhancement for Jailhouse Hypervisor on Multi-Core Platforms via Memory Isolation

Hubin Yang, Jiaming Zhang, Ruochen Shao, Yucong Chen, Rui Zhou*, and Qingguo Zhou*
Lanzhou University, Lanzhou, Gansu, China
{yanghb2019,jmzhang2020,shaorch21,chenyc18,zr,zhouqg}@lzu.edu.cn
*corresponding author

***Abstract*—With the improvement in computing power provided by modern multi-core architectures, software developers tend to integrate hybrid key systems into a single multi-core hardware platform to reduce costs. When integrating hybrid critical systems into a multi-core platform, critical tasks should not be affected by non critical tasks, so isolating shared resources between different critical levels is a major challenge.**

As virtualization technology can provide complete isolation of hardware shared resources, in this paper, a mechanism that prevents the DRAM bank interference from non real-time cell and hypervisor to real-time cell on multi-core platforms via bank partitioning to enhance real-time for the Jailhouse hypervisor is proposed. Based on that mechanism, the memory access of non real-time cell and Jailhouse hypervisor will not cause DRAM bank interference to all real-time tasks which run in the real-time cell and are limited to accessing the physical pages belonging to the DRAM bank assigned to the real-time cell. A series of experiments are performed on the Raspberry Pi 4 show that RJMM is effective in improving the real-time and isolation performance composed of the native Jailhouse.

Keywords*—*bank partitioning, Jailhouse, real-time, isolation, partitioning hypervisor

I. Introduction

With the increase of computational power and performance which are offered by modern multicore architectures, there is a trend that mixed-criticality systems, such as autonomous vehicles systems in which the real-time and non real-time tasks with different safety and timing requirements are integrated and scheduled, tend to be consolidated into a single multicore hardware platform by software developers to reduce the costs and improve the maintainability of software.

Although the demand for such consolidating is ever-growing, it causes a new problem stemming from the interference generated by the contention of shared resources. The DRAM is one sort of shared resources on modern hardware multicore platforms, and the DRAM memory sharing between all cores in system may cause memory contention and interference problems, leading to the performance degradation of real-time tasks due to the delays introduced by the memory contention.

In order to integrate a mixed-criticality system in which critical tasks should not be affected by the non-critical ones on a multicore platform with shared memory, the isolation for those shared resources among different criticality tasks is a significant challenge.

The virtualization technology applied in various fields is often regarded as a promising solution for such consolidation. Since the virtualization technology can provide the isolation of shared resources, with the help of it, it becomes possible to integrate tasks with different critical levels on the same physical platform. However, while some hypervisors, such as KVM and Xen, have been initially designed not to consider the isolation of shared sources which can cause contention and interference problems. Fortunately, there are some lightweight partitioning hypervisors oriented to real-time and safety-criticality domain, such as ACRN [1], Bao [2], Jailhouse [3] and XtratuM [4], which take into account the need of isolation.

Especially, A number of related studies [4][5][6] use the memory bank partitioning to reduce memory interference for improving performance in virtualization. Based on the MPKI and RBH of the VM, DCRM [5] partitions all VMs into three types and adopt different strategies for different types of VMs. VMMP [4] dynamically maps hypervisor, VMs and applications onto different memory banks on KVM hypervisor to improve performance in cloud computing, but it needs a lot of banks, which may be satisfied in cloud computing platforms, however the demand for so many banks on resource limited ones can not be met. What's more, when the number of memory banks is not enough, each two applications of one VM will share the same bank group.

To solve the above described issues, in this paper, we propose RJMM, a mechanism that prevents the DRAM bank interference from non real-time cell and hypervisor to real-time cell on multi-core platforms via bank partitioning, to enhance real-time for the Jailhouse hypervisor. When creating a cell, RJMM maps the cell into specified banks according to whether or not it is a real-time cell. To prevents the bank interference from hypervisor to real-time cell, RJMM maps the memory allocated by Jailhouse hypervisor into specified banks different from that of real-time cell. Therefore, the accesses to memory by the non real-time cell and Jailhouse hypervisor will not cause DRAM bank interference to all real-time tasks running in the real-time cell which limited to accessing physical pages belonging to the DRAM banks allocated to the real-time cell.

We use the Cyclictest and Isol-Bench benchmarks to conduct a series of experiments on our experimental platform to assess the real-time and isolation performance achieved by our mechanism, compared to the native Jailhouse. The evaluation results demonstrate that our mechanism is effective in improving the real-time and isolation performance in virtualization. The results with the Cyclictest benchmark show that the RJMM performs good real-time capability compared with the native Jailhouse. Especially, the average and maximum latency of RJMM respectively went down by 73.7% and 73.5% compared to the native Jailhouse. In the Isol-Bench benchmark experiment, RJMM increases the running speed of Isol-Bench by an average of 1.5% compared with the native Jailhouse.

The main contributions of this paper are summarized as follows:

1. We propose a new mechanism which prevents the DRAM bank interference from non real-time cell and hypervisor to real-time cell on multi-core platforms via bank partitioning to enhance real-time for the Jailhouse hypervisor
2. We implement our mechanism in Jailhouse 0.12 for evaluation.
3. We perform a series of experiments to assess the real-time and isolation performance achieved by our mechanism compared to the native Jailhouse on our experimental platform.

The rest of this paper is organized as follows. Section II provides related works. The design and implementation of our mechanism are provided in Section III. Section IV describes the evaluation results including the experimental setup. Finally, we conclude in Section V.

II. Related Work

In this section, we review some of the recent related works about lightweight partitioning hypervisors and DRAM bank partitioning for hypervisor.

A. Lightweight Partitioning Hypervisor

There are some lightweight partitioning hypervisors thar are oriented to real-time and safety-criticality domains, such as ACRN [1], Bao [2], Jailhouse [3] and XtratuM [4]. ACRN is a lightweight hypervisor developed by Intel on x86 architecture for the Internet of things, which meets low-latency access requirements and relies on Linux to boot the system. XtratuM is designed to satisfy the highly critical requirements in the aerospace, which follows the ARINC 653 standard, such as considering temporal and spatial isolation, low overhead, efficient inter-partition communication and so on. The resources allocated to each partition are configured by a configuration file. Bao is a lightweight static partitioning hypervisor, which is implemented on ARM and RISC-V platforms and does not rely on Linux. It provides a transparent and secure partitioning layer for critical situations. Jailhouse performances the low overhead and low latency. Like ACRN, Jailhouse also relies on the Linux boot system. Some projects are relying on the spatial isolation capability of Jailhouse to isolate real-time and non real-time systems, such as piCASSO, Hercules, RETINA and SELENE project [7][8][9][10].

B. Bank Partitioning in Virtualization

Several related studies [4][5][6] use the memory bank partitioning to reduce memory interference for improving performance in virtualization. VMMP [4] implemented in KVM hypervisor maps hypervisor, VMs and applications into different memory banks, the strategy used by VMMP is that (1) when the number of banks is enough, the KVM hypervisor and each application of one VM are allocated 16 banks respectively and the remaining banks are allocated to all VMs, (2) when it is not enough, the KVM hypervisor and all VMs are allocated 16 banks respectively and each two applications of one VMs will share a bank group. Based on the MPKI and RBH of the VM, DCRM [5] divides all VMs into three types and allocates memory banks according to the VM type. The bank allocation strategy of DCRM is that 16 unique memory banks are allocated for memory-bound VMs with high row buffer locality, the memory-bound VMs with low row buffer locality are allocated one of 16 bank groups and CPU-bound VMs can use all memory banks.

III. Design and Implementation

RJMM is a mechanism that prevents the DRAM bank interference from non real-time cell and hypervisor to real-time cell on multi-core platforms via bank partitioning, to enhance real-time for the Jailhouse hypervisor. In this section, we present the design and implementation of RJMM.

The design of RJMM includes two key techniques: (1) when creating a cell, RJMM can map the cell into specified banks according to whether or not it is a real-time cell (see Section 3.2). (2) To prevents the bank interference from hypervisor to real-time cell, RJMM map the memory allocated by Jailhouse hypervisor into specified banks different from those of real-time cells (see Section 3.3).

As shown in Figure 1, when creating cells, we assign bank group 1 to the real-time cell and bank group 0 to non real-time cell and Jailhouse hypervisor. It is implemented during cell create phase to allocate bank group 0 and bank group 1 to the real-time cell and the non real-time cell separately, and allocating bank group 0 to Jailhouse hypervisor is achieved in the enable stage. In this way, all non real-time tasks running in a non real-time cell and Jailhouse hypervisor are limited to access physical pages belonging to the DRAM banks not allocated to real-time cell, so that the execution of real-time tasks will not be affected by that of non real-time tasks and hypervisor.

A. DRAM Bank Mapping

In order to allocate a page belonging to a specified DRAM bank to a specified cell, we must first obtain the DRAM address mapping information from physical address to DRAM address that can determine which DRAM bank a physical page belongs to. At present, some studies [11][12] use reverse engineering to obtain DRAM address mapping information. These

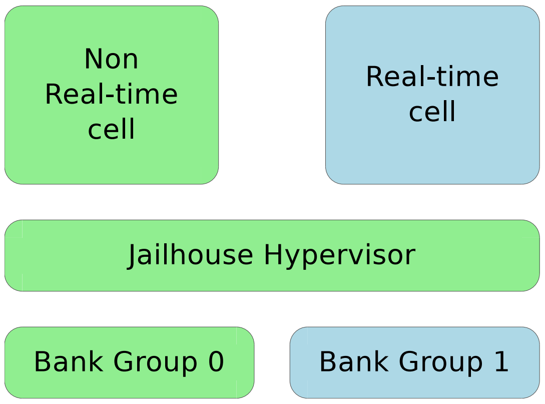

Fig. 1. Assigning the specified bank group to the specified cell and Jailhouse Hypervisor.

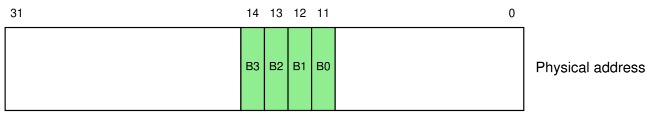

Fig. 2. The memory mapping information of our experimental platform.

reverse engineering solutions are aimed at those architectures which do not reveal DRAM address mapping information in their official manuals, such as Intel Architecture. In this work, we use Raspberry Pi 4 as our experimental platform to evaluate RJMM and we can get the mapping information of Raspberry Pi 4 from [13].

Figure 2 illustrates the mapping information of the DRAM Bank of the experimental platform in this paper. It can be seen from the figure that the four bank bits are determined by bits 11 to 14 of the physical memory address. So we can divide all 16 banks into two groups and assign bank groups 0 to the non real-time cell and Jailhouse hypervisor, bank groups 1 to the real-time cell.

B. Allocating Bank Group to Cell

There are usually two address translation stages in virtualization scenario. In the first stage, the virtual address of VM called Guest Virtual Address (GVA) is translated to the physical address of VM called Guest Physical Address (GPA), at second stage, the GPA needs to be translated to Host Physical Address (HPA) in which the data required is stored.

Jailhouse Hypervisor allocates memory for a cell depending on the configuration in the cell configuration file in which there are some memory regions defined by a virtual address space and a physical address space, both of which have the same size. At the create stage, Jailhouse creates mapping between GPA (the address within the virtual address space of the memory region) and HPA (the address within the physical address space of the memory region) for cells, so that when cells are running, the GPA of cell will be mapped to HPA by the hardware address translation unit.

To map a cell into specified banks, RJMM modifies the memory region in the cell configuration file to be defined by a virtual address space and the size of that. During the cell creating phase, RJMM creates mapping between GPA (the address within the virtual address space of the memory region) and HPA (the address within the memory bank indicated by the identifier of the cell) for the cell according to the modified cell configuration file. Consequently, the image of the cell will be loaded into the designated bank group at the following cell load stage, and the memory allocated to the processes running in the cell is in the bank group during runtime.

C. Allocating Bank Group to Jailhouse

Jailhouse Hypervisor maintains a memory pool called mem_pool from which Jailhouse allocates free pages and to which it frees pages during runtime. To prevents the bank interference from hypervisor to real-time cell, RJMM map the memory allocated by Jailhouse hypervisor into specified banks different from those of real-time cells. This is done by making all free pages in mem_pool belong to the bank group 0 in the enable stage, so that the memory allocated by Jailhouse belong to bank group 0 during runtime.

IV. Evaluation

In this section, we describe the experimental platform used in our experiments, and then a series of experiments have been performed on the Raspberry Pi 4 to assess the performance isolation and real-time achieved by our proposed solutions. All experiments are repeated three times, and their averages are used.

TABLE I
Experimental environment

Parameter	Configuration
Processor	ARM Cortex-A72
	Quad Core Processor 1.5GHz
L1 I-cache	48KB
L1 D-cache	32KB
L2 shared cache	1MB
DRAM Memory	4GB LPDDR4 SDRAM
	16 banks

A. Experimental Setup

Table I lists the major parameters of the processor and DRAM memory of Raspberry Pi 4 used in this paper. As shown in the table, the Raspberry Pi 4 uses a quad-core ARM Cortex-A72 processor with 32 KB L1 Data cache, 48 KB L1 Instructions cache and 1 MB L2 cache. As mentioned in section 3.1, we divide all 16 banks into two groups, each bank group contains 8 banks, namely 2GB memory.

We used Jailhouse 0.12 to create two cells, the real-time cell used for measuring and the non real-time cell used to generate interference. Each cell is assigned one core, and both cells host Linux 5.4. The evaluation focuses on comparing the native Jailhouse with the modified one integrating the isolation capabilities.

Cyclictest is part of the rt-tests and is designed to measure real-time performance. The Isol-Bench benchmark [14] consists in accessing a portion of memory with a size of N KB

TABLE II
ADDRESS MAPPING OVERHEAD

Size (MB)	the native (ms)	RJMM (ms)	Increase
1	0.0648	0.0938	45%
4	0.2441	0.3532	45%
16	1.0186	1.4387	41%
64	4.0682	5.7532	41%
256	16.2878	23.0213	41%

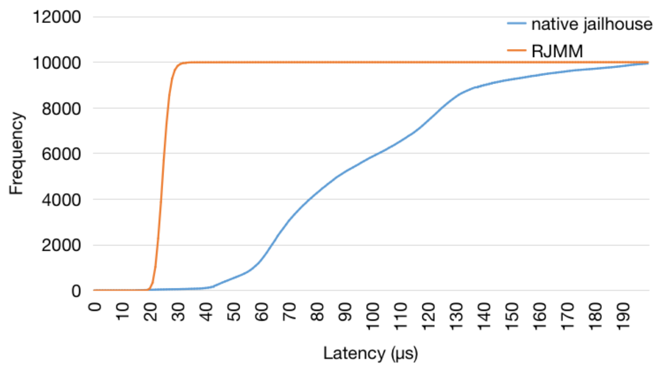

Fig. 3. The cumulative distribution function (CDF) of the frequency.

for M times in a sequential manner. we use it to evaluate the isolation performance of RJMM.

B. Implementation Overhead

Compared with unmodified Jailhouse, the overheads introduced by RJMM comes from the creating of address mapping for cells during the cell creating phase. To evaluate the introduced overhead of RJMM, we measure the total time to complete the address mappings of memory regions of different sizes.

Table II shows the address mapping times of the native Jailhouse and RJMM during the phase of creating a cell, with the sizes of address regions varying from 1 MB to 256 MB with steps of 4 MB.

We can find from the table that although the address mapping times of RJMM is more than 40% of that of the native Jailhouse, It is worth noting that the mapping of GPA to HPA occurs only once during the lifetime of a cell. Therefore, once the address mapping of a cell has been accomplished, RJMM will not lead to any runtime overhead to the cell.

C. Real-time

In this experiment, we make use of Cyclictest as benchmark to conduct real-time evaluation of RJMM [15]. The non real-time cell is configured to continuously access 16 MB of memory to generate inter-cell memory interference, and then We run Cyclictest on the real-time cell to compare the latency of the native and modified Jailhouse hypervisor.

The cumulative distribution function (CDF) of the frequency is illustrated in Figure 3, and Figure 4 shows the normalized frequencies of three types under the native Jailhouse and RJMM (normalized to the native Jailhouse). We can see from both figures that the latency of RJMM is less than that of the native Jailhouse. Although the minimum latencies of RJMM and the native Jailhouse make no difference, the average and maximum latency of RJMM respectively went down by 73.7% and 73.5% compared to the native Jailhouse. It illustrates that the RJMM achieves good real-time capability compared with the native Jailhouse.

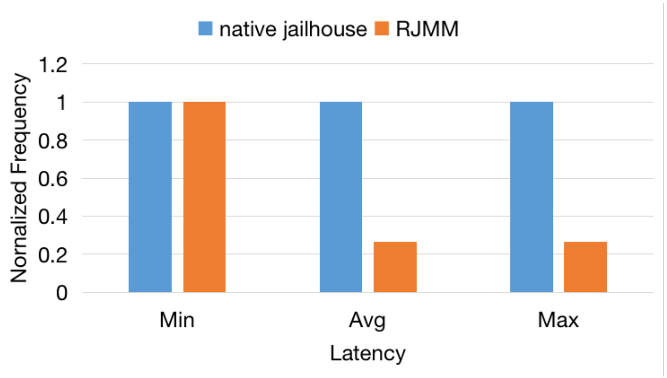

Fig. 4. The normalized frequencies.

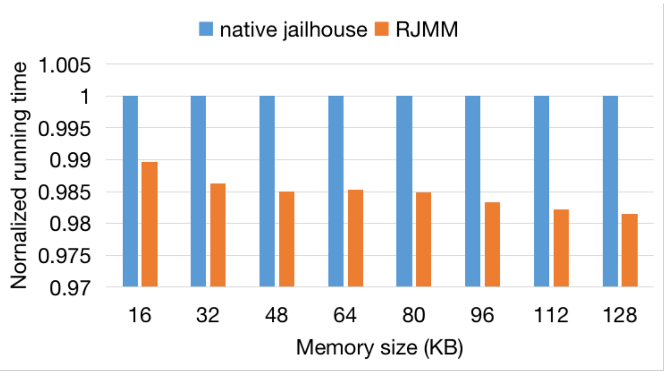

Fig. 5. The normalized running times.

D. Isolation Performance

In this experiment, we use Isol-Bench benchmark to investigate the isolation performance of RJMM [14]. Like the previous experiment, the non real-time cell continuously access 16 MB of memory to generate inter-cell memory interference, and then run the Isol-Bench on the real-time cell to measure the memory access time. With the size N of the memory varying from 16 KB to 128 KB with steps of 16 KB and the times M of execution being 10000.

The results are shown in Figure 5, where the x-axis represents the size N and the y-axis shows the running time normalized to the case where the Isol-Bench runs under the native Jailhouse. From the figure, we can easily see that the RJMM performs better. Compared with the native Jailhouse, RJMM has increased the running speed of Isol-Bench by an average of 1.5%.

V. CONCLUSION

To consolidate a mixed-criticality system into a single multicore platform, the isolation for shared resources among different criticality tasks is a significant challenge. In this paper, we propose RJMM, a mechanism that prevents the DRAM bank interference from non real-time cell and hypervisor to real-time cell on multi-core platforms via bank partitioning to enhance real-time for the Jailhouse hypervisor. We conducted a series of experiments on our experimental platform to evaluate the real-time and isolation performance of our mechanism. The evaluation results show that our mechanism is effective in improving the real-time and isolation performance of virtualization. Especially, the results with the Cyclictest benchmark show that the RJMM performs good real-time capability, and the average and maximum latency of RJMM respectively went down by 73.7% and 73.5% compared with the native Jailhouse.

In our current prototype, RJMM only considers DRAM bank partitioning, which limits the effectiveness of solving the interference problems. We believe that Integrating memory bandwidth reservation [14] and cache partitioning [16] will help to provide better isolation performance, which is still our future work.

Acknowledgment

This work was partially supported by National Key R&D Program of China under Grant No. 2020YFC0832500, the Fundamental Research Funds for the Central Universities under Grant No. lzujbky-2021-sp57 and National Natural Science Foundation of China under Grant No. 61402210.

References

[1] H. Li, X. Xu, J. Ren, and Y. Dong, "ACRN: a big little hypervisor for iot development," in *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE 2019, Providence, RI, USA, April 14, 2019*, J. B. Sartor, M. Naik, and C. Rossbach, Eds. ACM, 2019, pp. 31–44.

[2] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems," in *NG-RES 2020*, 2020.

[3] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer, "Look mum, no VM exits! (almost)," *CoRR*, vol. abs/1705.06932, 2017.

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

[5] G. Jia, G. Han, J. Jiang, S. Chan, and Y. Liu, "Dynamic cloud resource management for efficient media applications in mobile computing environments," *Personal and Ubiquitous Computing*, 2018.

[6] T. Kloda, M. Solieri, R. Mancuso, N. Capodieci, and M. Bertogna, "Deterministic memory hierarchy and virtualization for modern multi-core embedded systems," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

[7] C. Hernandez, J. Flieh, R. Paredes, C. A. Lefebvre, and M. Labayen, "Selene: Self-monitored dependable platform for high-performance safety-critical systems," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, 2020.

[8] P. Burgio, M. Bertogna, I. S. Olmedo, P. Gai, and M. Sojka, "A software stack for next-generation automotive systems on many-core heterogeneous platforms," in *2016 Euromicro Conference on Digital System Design (DSD)*, 2016.

[9] A. Db, A. Pg, B. Fs, C. Cea, B. Jb, and A. Jf, "Modular smart controller for industry 4.0 functions in machine tools," *Procedia CIRP*, vol. 81, pp. 1331–1336, 2019.

[10] A. Biondi, M. Marinoni, G. Buttazzo, C. Scordino, and P. Gai, "Challenges in virtualizing safety-critical cyber-physical systems," in *Proceedings of Embedded World Conference 2018*, 2018, pp. 1–5.

[11] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "Dramdig: A knowledge-assisted tool to uncover DRAM address mapping," in *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. IEEE, 2020, pp. 1–6.

[12] C. Helm, S. Akiyama, and K. Taura, "Reliable reverse engineering of intel dram addressing using performance counters," in *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2020.

[13] M. G. Bechtel and H. Yun, "Memory-aware denial-of-service attacks on shared cache in multicore real-time systems," *CoRR*, vol. abs/2005.10864, 2020.

[14] P. Modica, A. Biondi, G. C. Buttazzo, and A. Patel, "Supporting temporal and spatial isolation in a hypervisor for ARM multicore platforms," in *IEEE International Conference on Industrial Technology, ICIT 2018, Lyon, France, February 20-22, 2018*. IEEE, 2018, pp. 1651–1657.

[15] R. Ma, W. Ye, A. Liang, H. Guan, and J. Li, "Cache isolation for virtualization of mixed general-purpose and real-time systems," *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1405–1413, 2013.

[16] N. Suzuki, H. Kim, D. D. Niz, B. Andersson, and R. Rajkumar, "Coordinated bank and cache coloring for temporal protection of memory accesses," in *2013 IEEE 16th International Conference on Computational Science and Engineering (CSE)*, 2013.