Hot-patching Platform for Executable and Linkable Format Binary Application for System Resilience

Haegeon Jeong
Dept. of Computer Science and Engineering
Hanyang University, Korea
haegeonj@hanyang.ac.kr

Jinsung An*
Dept. of Computer Science and Engineering
Hanyang University, Korea
anjinsung@hanyang.ac.kr

Kyungtae Kang
Dept. of Computer Science and Engineering
Hanyang University, Korea
ktkang@hanyang.ac.kr

ABSTRACT

In modern society, software is ubiquitous and very complex and diverse. Examples of complex modern software include medical systems, flight systems, and high-reliability systems. A resilient system means a system that can continue to maintain or recover its function even if a problem occurs in the system. Elemental technologies of the resilience system include resistance, detection, reaction, and recovery. The combination of each technology constitutes a resilience system. In this paper, a hot-patching tool was designed and implemented as a recovery technique. Hot-patching is a technology that allows new features to be added and deleted without restarting the application. Based on this, an imprecise patch image was created and it was verified whether the application's computing resource usage could be dynamically adjusted using hot-patching. Through the experimental results, applicable mixed-criticality system example scenarios were described.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

Hot-patching, ELF, Linux, binary, non-stop service, system resilience

ACM Reference Format:
Haegeon Jeong, Jinsung An, and Kyungtae Kang. 2023. Hot-patching Platform for Executable and Linkable Format Binary Application for System Resilience. In *Proceedings of ACM SAC Conference (SAC'23).* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3555776.3577821

1 INTRODUCTION

In modern society, software is ubiquitous and very complex and diverse. Examples of complex modern software include medical systems, flight systems, and high-reliability systems. Such software must have safety thresholds and must not be interrupted, and must be designed considering high performance and high resilience even in situations such as disasters. However, even if a program planned with the best quality is written, damage to the system due to unpredictable external events is unavoidable.

Cyber-physical system (CPS) is a next-generation system that combines computing, communication, and control technologies. It must satisfy performance levels while satisfying stability, reliability, robustness, and resilience. In particular, resilience is becoming important in CPS design because many social infrastructures are built with CPS[6]. Resilience refers to a system that can endure tolerate a certain level of external attack or internal faults by either partially operating normally or gradually shutting down the system. Figure 1 shows how the resilience technique is applied in CPS. Resistance resilience can be used as a way to isolate the system from external factors. When external influences invade the resistance resilience and begin to affect the system, detection resilience can detect it, and reaction resilience can mitigate the failure before the system fails. Finally, system resilience is to reduce system failure to a tolerable level through recovery. Resiliency technologies are abstract and must be implemented in a system to achieve the intended effect. However, if a technology is chosen or implemented incorrectly, the results may not be as intended and the system may be less resilient.

Imprecise computing has emerged as an approach for designing energy-efficient digital systems [2]. It results in some loss of accuracy in the calculated results and is applicable to many systems and applications. It can significantly improve energy efficiency through approximation techniques by relaxing accuracy for fully accurate or completely deterministic tasks. Imprecise computing can also benefit real-time systems. For example, in a virtual reality tracking system, violating deadlines can result in stuttering of the video, impeding mission success. In contrast, errors that occur in a few pixels out of a small number of frames are usually not discernible to the human eye and have far less serious consequences than a

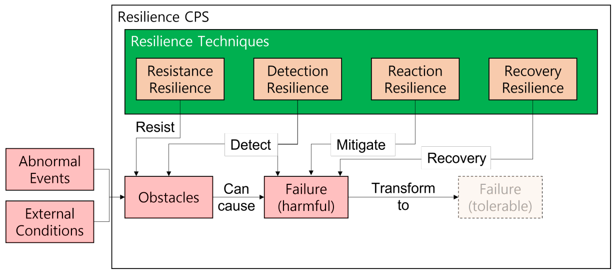


Figure 1: Resilience technique work flow in cps system.

*Major in Bio Artificial Intelligence

SAC'23, March 27 –March 31, 2023, Tallinn, Estonia
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-9517-5/23/03…$15.00
https://doi.org/10.1145/3555776.3577821

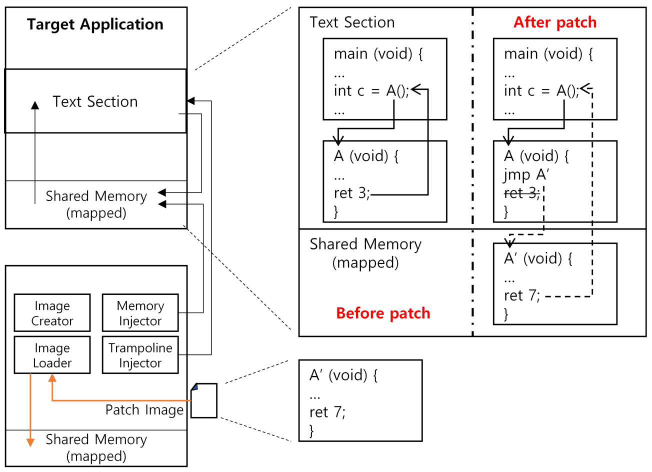


Figure 2: The hot-patcher overall structure.

deadline violation. Imprecise computing allows relatively short execution times, which can be adopted to avoid deadline violations when computing resources are stressed [3].

There is a functional-level hot-patching platform (FLHP) available for uninterrupted service[4]. The FLHP is a platform that considers the functions of applications as modules and patches these modules. This platform can perform updates on a function-by-function basis without interrupting existing applications that are currently running and can update the debugging versions of functions that produce errors. In this paper, we proposed a patch image structure by improving FLHP. And it was confirmed that the consumption of computing resources can be reduced by using the PI value calculation and image compression algorithm, and based on this, a mixed-criticality system (MCS) scenario is proposed.

2 HOT-PATCHING IMPLEMENTATION

We implemented a hot-patching tool and patch image considering portability, usability, and expandability. These features are designed to modularize the necessary functions such that only the relevant modules are engaged according to the current situation. To use the proposed hot-patching platform, users must install GCC version 4.7 or later. We have minimized dependency on other applications since there are no other requirements outside of compiler restrictions. This facilitates support across multiple platforms.

As shown in Figure 2, the proposed hot-patching platform consists of four independent modules. Each module is completely independent of target processes. The modules are largely divided into modules that handle patch images and modules that manipulate target processes. The modules that handle patch images are the image creator and image loader. These modules create a patch image with a specific name and load it into memory. The modules that manipulate target processes are the memory injector and trampoline injector modules, both of which insert the functionality required by the hot-patcher into the target process. The figure drawn on the right in Figure 2 shows the target application before and after hot-patching. As a result, when function A is called, actual patching is performed by jumping to function A'.

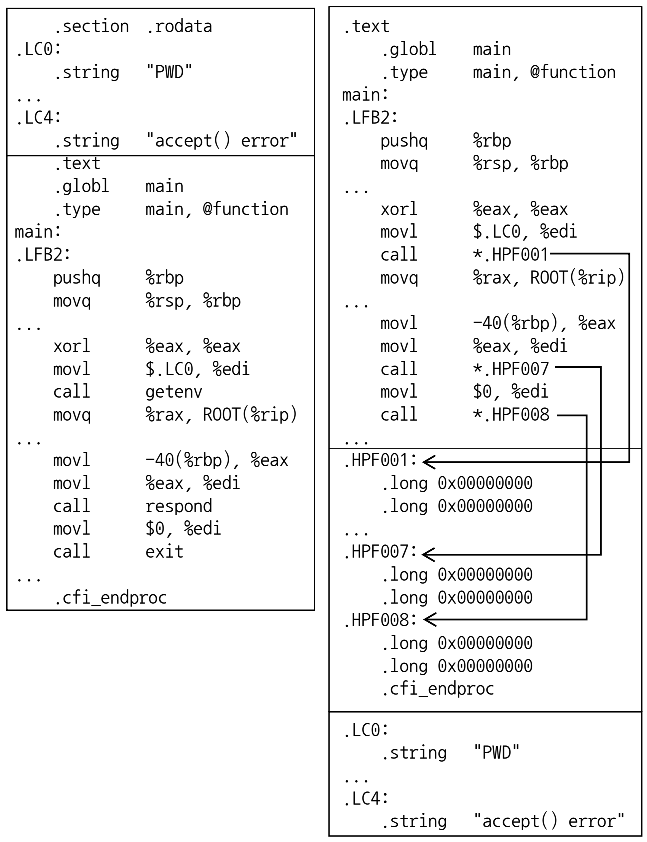
The patch image structure is designed to facilitate hot-patching. Figure 3 presents the assembly code before and after modification. The left-hand side presents the original assembly file from the compilation stage of a specific application. The structure of the assembly file is divided into a rodata section containing a static object and text area containing the actual execution code. Listing 1 attaches part of the script to create the structure shown in the figure. Through this, it is possible to easily link the address of a function or data during hot-patching.

```
PADDING_FORMAT = \
    '%s:\t.long 0x00000000\n\t.long 0x00000000\n'
with open('original_source.s', 'r') as f:
    data = f.readlines()
    ref_index = 0;

    for idx, line in enumerate(data):
        if '@function' in line:
            CF = line.split('\t')[2].split(',')[0]
            FCMD[CF] = []

        if 'call' in line:
            if CF is None: # not exists functions
                exit(-1)

            elements = line.split('\t')
            function_name = elements[2].strip()
            call_index = '.HPF%03d'%(HPF_NUMBERING,)
            elements[2] = '*%s\n'%(call_index, )
            HPF_NUMBERING = HPF_NUMBERING + 1
            line = '\t'.join(elements)
            data[idx] = line

            FCMD[CF].append(
                {'function_name': function_name, \
                 'call_index': call_index,\
                 'ref_index':ref_index})
            ref_index = ref_index + 1

        if 'cfi_endproc' in line:
            ref_index = 0
            relocation_space = ''
            for meta_data in FCMD[CF]:
                relocation_space = relocation_space\
                                   + PADDING_FORMAT
                %(meta_data['call_index'], )
            data[idx] = relocation_space+line
    m_file = open('modified_source.s', 'w')
    m_file.write(''.join(data))
    m_file.close()
```

Listing 1: Modifying original source code

The trampoline injector is a component that modifies workflows by inserting jump instructions at the entry points of original functions. Intel CPUs are designed using a complex instruction set computer (CISC) model. The CISC model allows direct jumps to any address because the length of instructions is variable. In contrast, in ARMv7 processors, the length of instructions is fixed at 4 bytes. This means the jump instructions in an ARM processor operate based on relative addresses. Therefore, to cover the entire managed memory area in an ARM processor, CPU registers must be used.

Listing 2 presents the code for generating trampoline code on an ARM-based device. This is a machine code generator that accepts the start address of a patched function as an argument. The address is stored in the r12 register and loaded into the PC register. In



```
    .section  .rodata
.LC0:
    .string   "PWD"
...
.LC4:
    .string   "accept() error"
    .text
    .globl    main
    .type     main, @function
main:
.LFB2:
    pushq     %rbp
    movq      %rsp, %rbp
...
    xorl      %eax, %eax
    movl      $.LC0, %edi
    call      getenv
    movq      %rax, ROOT(%rip)
...
    movl      -40(%rbp), %eax
    movl      %eax, %edi
    call      respond
    movl      $0, %edi
    call      exit
...
    .cfi_endproc
```

```
.text
    .globl    main
    .type     main, @function
main:
.LFB2:
    pushq     %rbp
    movq      %rsp, %rbp
...
    xorl      %eax, %eax
    movl      $.LC0, %edi
    call      *.HPF001
    movq      %rax, ROOT(%rip)
...
    movl      -40(%rbp), %eax
    movl      %eax, %edi
    call      *.HPF007
    movl      $0, %edi
    call      *.HPF008
...
.HPF001:
    .long 0x00000000
    .long 0x00000000
...
.HPF007:
    .long 0x00000000
    .long 0x00000000
.HPF008:
    .long 0x00000000
    .long 0x00000000
    .cfi_endproc
.LC0:
    .string   "PWD"
...
.LC4:
    .string   "accept() error"
```

Figure 3: Assembly source code before and after modification.

contrast, on x86_64-based devices, one can generate trampoline code using only a single 12 byte instruction. The generated code is inserted at the function entry point using the ptrace system call. This enables jumping to patched functions through the trampoline code, even if original code is executed.

```
void get_trampoline_code(unsigned long target_addr,
     unsigned long *code) {
    unsigned long upper = (target_addr & 0xffff0000)>>16;
    unsigned long lower = target_addr & 0xffff;
    unsigned long movt_ins = 0xe340c000;//movt r12, #0
     xXXXX
    unsigned long orr_ins_8__15 = 0xe38ccc00;
    unsigned long orr_ins_0__8 = 0xe38cc000;
    /* above 3 instructions set the r12 register
    to shared memory starting target_addr */
    unsigned long set_pc_to_target = 0xe1a0f00c;
    movt_ins |= (upper & 0x0fff);
    movt_ins |= ((upper & 0xf000) << 4);
    orr_ins_8__15 |= ((lower & 0xff00) >> 8);
    orr_ins_0__8 |= lower & 0x00ff;
    *code = movt_ins; *(code+1) = orr_ins_8__15;
    *(code+2) = orr_ins_0__8; *(code+3) =
     set_pc_to_target;
}
```

Listing 2: Trampoline code generator for ARM processor (Raspberry Pi).

3 EXPERIMENT

To test the functionality of hot-patching, we implemented an application that calculates the PI value and imprecise patched it. According to Bailey and Borwein, when calculating with high precision in mathematical physics, the number of significant digits is considered to be 17 digits [1]. To show that imprecise computing can be performed through hot-patching in general applications, we tested an application that calculates the PI value. We conducted experiments based on 15 significant digits.

Figure 4 shows the results of the reduction in resource usage based on patches that reduce the accuracy of any computational application. This experiment aims to reduce CPU usage instead of sacrificing acceptable accuracy via hot-patching and changing the applications to be run. The figure shows the results of measuring the CPU resources used by an application calculating the PI with hot-patching. The result was obtained by performing an imprecise patch that changes the number of significant digits through hot-patching. Based on 15 significant digits, the consumption of computing resources decreased significantly up to 10 digits, and it was almost the same level below 10 digits. Through this result, it was confirmed that the normal operation of hot-patching and the application of imprecise patch can save computing resources.

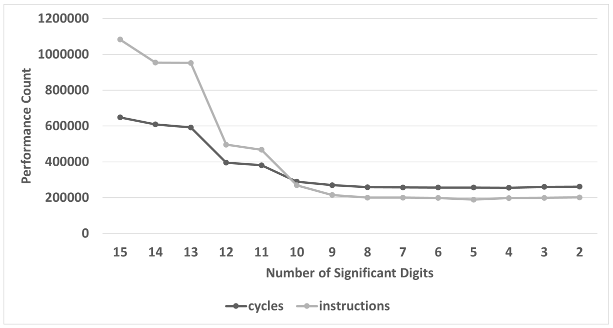


Figure 4: Changes in cycles and instructions according to the number of significant digits of the PI value.

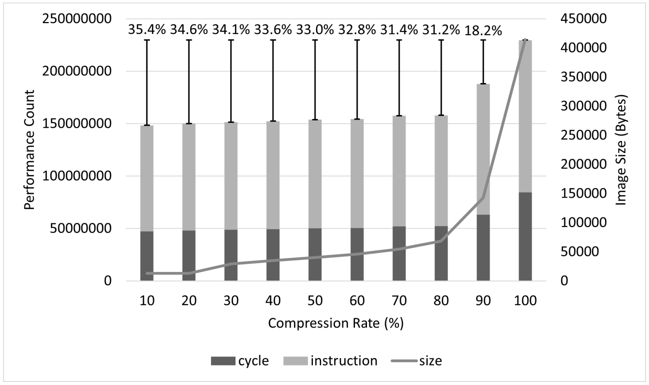


Figure 5: Changes in consumption of computing resources according to compression ratio.

4 EVALUATION

As a result of the experiment through imprecise patch, it was confirmed that resources can be saved without terminating the program while patching the accuracy of the PI significant value finding algorithm. In addition to this experiment, we adjusted the compression rate while converting png images to jpeg to see if resource consumption could be reduced at the expense of accuracy. Figure 5 expresses the tendency of computing resources to decrease according to the compression rate. As can be seen from the chart, at a compression rate of 80%, the CPU resource was reduced by 31.2% and the image size was reduced by about 83%. It can be seen that the resource consumption of the system can be reduced through various algorithms in a situation where resources are insufficient in the system.

Using the previously implemented hot-patching tool, an imprecise scenario for system resilience can be established. Assume a situation in which the resource usage of the system rapidly increases due to an external factor or the core is broken due to an issue such as over current. Also, as shown in the Figure 6, the system has tasks from T1 to T4. Among them, it is assumed that T1 to T3 are relatively unimportant tasks. If an emergency occurs, reduce the CPU resource usage of each task by hot-patching the pre-made imprecise patches from T1 to T3. The increased CPU resources are allocated to the relatively more important T4 process to deal with emergencies. Then, after the emergency situation passes by using any method, remove the existing imprecise patch to return to the normal process so that the mission can continue.

In addition to the above scenarios, analyze application to identify statements that are not frequently executed. After that, after making the corresponding statement into a patch image, we can think of a way to patch and use it in real time when the statement is executed. As such, the hot-patching tool has the advantage of being able to show various effects depending on the input algorithm.

5 CONCLUSION

This paper proposes the development of hot-patching for building a resilient system. The resilience system is largely divided into

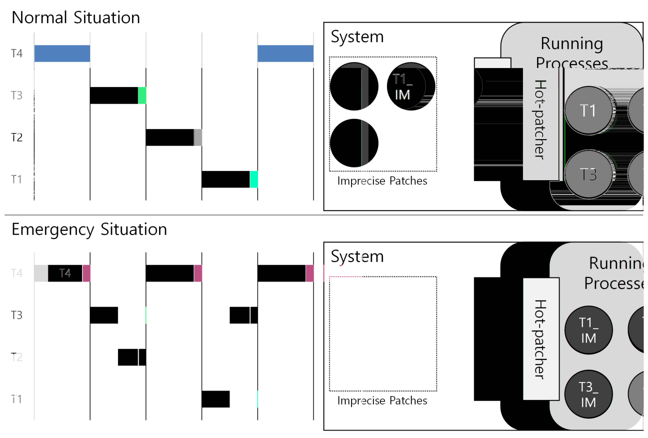


Figure 6: Imprecise computing scenario in mixed-criticality system.

hardware and software resilience systems. The hot-patching platform for ELF binary applications designed and implemented on both x86_64 and ARM architectures. The hot-patching platform supports function-based patching, meaning users can easily add or delete functionality. Additionally, because the proposed platform is independent of target processes, its operation does not affect existing applications, even if the platform experiences an issue. Based on experimental results, we determined that additional resources can be allocated by reducing accuracy. By extending these experimental results, the degree of saving of computing resources according to the compression rate during image conversion was examined. And based on this, imprecise mixed criticality scheduling scenario was examined. In another scenario, how to build an application without a specific algorithm and use it by hot-patching when needed. This means that even if the application is hijacked, it can be used as a security element that cannot see the content. As such, the hot-patching tool can show various effects depending on the input algorithm, and it can be felt that it is the same as stem cells.

In future, real-time MCS can be built with the proposed hot-patching technology. The system recovery method in general MCS is task dropping. Task dropping is to prevent system faults by removing relatively less important tasks from the scheduler [5]. Using the hot-patching proposed in this paper, it will be possible to build a more flexible system than the rigid task dropping algorithm by applying imprecise computing. As a practical method for implementation, it can be implemented with raspberry pi, preempt-rt patch, and scheduling tuning.

ACKNOWLEDGMENTS

This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No.RS-2022-00155885, Artificial Intelligence Convergence Innovation Human Resources Development (Hanyang University ERICA)) and the MSIT, Korea, under the Grand Information Technology Research Center support program (IITP-2022-2020-0-101741).

REFERENCES

[1] David H. Bailey and Jonathan M. Borwein. 2015. High-Precision Arithmetic in Mathematical Physics. *Mathematics* 3, 2 (2015), 337–367. https://doi.org/10.3390/math3020337
[2] Jie Han and Michael Orshansky. 2013. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*. 1–6. https://doi.org/10.1109/ETS.2013.6569370
[3] Lin Huang, Youmeng Li, Sachin S. Sapatnekar, and Jiang Hu. 2018. Using Imprecise Computing for Improved Non-Preemptive Real-Time Scheduling. In *Proceedings of the 55th Annual Design Automation Conference (DAC '18)*. Association for Computing Machinery, New York, NY, USA, Article 71, 6 pages. https://doi.org/10.1145/3195970.3196134
[4] Haegeon Jeong, Jeanseong Baik, and Kyungtae Kang. 2017. Functional level hot-patching platform for executable and linkable format binaries. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 489–494. https://doi.org/10.1109/SMC.2017.8122653
[5] Jaewoo Lee, Hoon Sung Chwa, Linh T. X. Phan, Insik Shin, and Insup Lee. 2017. MC-ADAPT: Adaptive Task Dropping in Mixed-Criticality Scheduling. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 163 (sep 2017), 21 pages. https://doi.org/10.1145/3126498
[6] Yuchang Won, Buyeon Yu, Jaegeun Park, In-Hee Park, Haegeon Jeong, Jeanseong Baik, Kyungtae Kang, Insup Lee, Sang Hyuk Son, Kyung-Joon Park, and Yongsoon Eun. 2018. An Attack-Resilient CPS Architecture for Hierarchical Control: A Case Study on Train Control Systems. *Computer* 51, 11 (2018), 46–55. https://doi.org/10.1109/MC.2018.2876054