Smart Media Journal / Vol.14, No.2 / ISSN:2287-1322
https://dx.doi.org/10.30693/SMJ.2025.14.2.26

BMC 성능 최적화를 위한 커널 코드 변경의 효율적인 관리

(Efficient Management of Kernel Source Code Changes for Optimizing BMC Performance)

오경은*, 김한결**, 오정환**, 안성모**, 홍지만***
(Kyeong Eun Oh, Han Gyeol Kim, Jeong Hwan Oh, Seong Mo An, Ji Man Hong)

요 약

클라우드 및 대규모 데이터 센터의 증가로 서버 유지 비용이 증가하면서, 효율적인 원격 모니터링 시스템에 대한 수요가 커지고 있다. 베이스보드 관리 컨트롤러(BMC)는 서버 상태를 실시간으로 감시하여 문제를 조기에 감지하고 신속히 대응함으로써 다운타임을 최소화하고, 자원 활용을 최적화해 서버 인프라의 원활한 운영을 지원한다. 서버 장애 시 빠른 재부팅을 통해 데이터 손실을 방지하고 비즈니스 연속성을 유지하는 역할을 한다.

본 논문은 BMC의 빠른 재부팅을 위한 프로파일링 도구의 최적화 방법을 제안한다. 이를 위해, 기존 프로파일링 도구의 비효율적인 대규모 커널 소스 전송 문제를 해결하고, 변경된 파일만 전송하는 최적화 기법을 도입했다. 실험 결과, 본 방법이 데이터 전송 효율성을 높이고 시스템 자원 오버헤드를 줄이는 데 기여함을 확인했다.

■ 중심어 : 베이스보드 관리 컨트롤러 ; 임베디드 리눅스 ; diff 알고리즘 ; 최적화

Abstract

As server maintenance costs grow, the need for efficient remote monitoring rises. Baseboard Management Controllers (BMCs) help minimize downtime and optimize resources by monitoring server health and quickly rebooting after failures.

This paper presents an optimized BMC boot process, reducing resource overhead by transferring only modified files during boot, which improves data transfer efficiency.

■ keywords : BMC ; Embedded Linux ; Diff Algorithm ; Optimization

Ⅰ. 서 론

베이스보드 관리 컨트롤러(BMC)는 서버, 데이터 센터, 대규모 IT 인프라의 하드웨어 상태를 모니터링하고 관리하는 독립적인 마이크로 컨트롤러다. 서버의 메인 프로세서, 운영 체제, 메모리와 독립적으로 작동하여 서버가 꺼져 있어도 하드웨어 상태를 모니터링할 수 있다. 최근 클라우드와 대규모 데이터 센터의 수요 증가로 운영 비용, 특히 전력 소비와 같은 비용이 크게 증가했으며[1], 이를 절감하기 위해 BMC는 원격 모니터링 및 재부팅, 전원 제어가 가능해야 한다[2].

BMC는 온도, 전압, 팬 속도, 전원 상태 등의 하드웨어 상태를 실시간으로 감시하고 원격으로 관리할 수 있어 잠재적 문제를 조기에 발견하고 대응할 수 있다[4]. 이러한 원격 관리 기능은 데이터 센터의 운영 효율성을 높이는 데 필수적이다.

클라우드 기반 서비스와 대규모 데이터 센터에서는 수천에서 수만 대의 서버가 운영되며, 서버 중단이 비즈니스 연속성에 큰 영향을 줄 수 있다.[5]

BMC의 빠른 부팅 기능은 데이터 센터 운영자가 재부팅 시간을 단축하고 시스템 가동 시간을 최대로 유지하도록 돕는다. 특히 하드웨어나 소프트웨어 업데이트 이후 여러 서버를 동시에 재부팅해야 하거나 예기치 않은 시스템 장애가 발생할 때 신속한 복구가 가능해 서비스 품질을 유지할 수 있다[3].

* 준회원, 숭실대학교 컴퓨터학과
** 준회원, 숭실대학교 컴퓨터학부
*** 종신회원, 숭실대학교 컴퓨터학부
이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2022-0-00202).

접수일자 : 2024년 11월 25일
수정일자 : 2024년 12월 26일
게재확정일 : 2024년 12월 30일
교신저자 : 홍지만 e-mail : jiman@ssu.ac.kr

BMC의 빠른 부팅 기능을 위해서는 부팅 과정에서 발생하는 성능 병목을 식별하고 해결하는 프로파일링이 필요하다[6,7]. 이를 통해 시스템의 병목 지점을 파악하고 최적화된 부팅 절차를 설계할 수 있다. 이러한 프로파일링 도구는 부팅 시 CPU와 메모리 사용량, 부팅 시간, 커널 변경 사항을 추적 및 분석할 수 있어야 하며, 커널 소스 코드의 변경 사항과 해시 값을 기반으로 부팅 과정 중 발생하는 이벤트와 로그를 종합적으로 관리할 수 있다.

부팅 과정에서 생성되는 로그는 시스템 문제를 진단하는 데 필수적이다. 주요 부팅 메시지는 'dmesg.txt'에, 함수 호출 정보는 'trace.txt'에 기록된다. 빈번한 함수 호출로 인한 성능 저하 등의 문제 발생 시 로그 파일을 통해 최적화 방안을 찾을 수 있다. 이처럼, 프로파일링 도구는 커널 코드 변경 사항을 추적하고 시각화하는 기능을 포함하여 시스템의 안정성과 성능을 보장하며, 개발자가 시스템 내부 동작을 이해하고 효율적으로 최적화 작업을 수행하도록 돕는다.

본 논문에서는 BMC 환경에서 커널 소스 코드 변경 사항을 동적으로 시각화하고, 변경 로그를 효율적으로 관리 및 전송하는 방법을 제안한다. 제안된 방법은 대규모 커널 소스 코드 중 수정된 정보만 최소한으로 네트워크를 통해 전송하도록 설계되었다. 이를 위해 디렉터리 구조를 최적화하여 변경 로그의 크기를 줄이고, 수정된 파일의 경로와 해시 값을 통합하여 수정된 소스 코드 일부만 전송함으로써 커널 변경 로그를 효율적으로 비교할 수 있다. 수신된 데이터를 기반으로 커널 버전이 동적으로 추가 및 비교되며, 이에 따라 변경 사항이 시각화된다.

본 논문의 구성은 다음과 같다. 본론에서는 관련 연구와 그 한계점을 소개하고, 한계점을 보완하기 위한 커널 변경 로그 최적화 방법을 설명한다. 실험을 통해 제안된 효과를 입증한다. 마지막으로 논문을 결론짓는다.

Ⅱ. 본 론

1. 커널 변경 로그 최적화

'kernel-source_skin'은 커널 소스 코드에서 추가되거나 삭제된 파일을 구분하기 위해 디렉터리 구조와 빈 파일을 포함하는 디렉터리이다. 수정된 소스 코드를 식별하기 위해 'hash.txt'에 모든 파일의 해시 값을 저장하고, 'kernel-source_molt'에 수정된 파일의 디렉터리 구조만 복사한다. 이를 통해 커널 소스 코드 변경 사항을 보다 정확하게 추적할 수 있으며, 각 파일의 수정 내역을 쉽게 확인할 수 있다. [그림 1]에서는 이 과정을 설명한다.

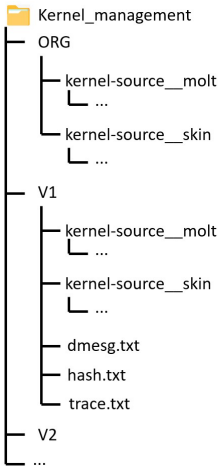

그림 1. 커널 버전 디렉터리 구조

이렇게 구성된 버전은 호스트 측에 저장되며, 사용자의 요청에 따라 비교를 위해 시각화된다. 커널 변경 로그의 경우, 각 버전의 추가 및 삭제 사항을 원본 커널 소스 코드와 비교하여 식별한 후, 후처리 및 시각화를 통해 두 버전 간의 변화를 비교한다. 이를 통해 개발자들은 커널의 발전 과정을 쉽게 추적하고, 특정 변경 사항이 시스템 성능에 미치는 영향을 보다 정확하게 분석할 수 있다.

커널 부팅 후 생성된 파일들은 현재 실행 중인 커널 및 빌드 버전의 정보를 동적으로 가져오도록 확장되어야 한다. 이 기능은 API를 통해 부팅 시간과 함수 호출 이력을 가져와 자동으로 버전을 생성할 수 있다. 그러나 커널 변경 로그 전송 중 여러 가지 문제가 발생할 수 있다. 특히, 실시간으로 커널 상태를 분석하려고 할 때 데이터 전송 지연 및 정확성 저하가 있을 수 있다. 따라서 커널 변경 로그를 효율적으로 전송하고 관리하기 위한 새로운 방법론이 필요하다.

커널 빌드 중 전체 커널 소스 코드를 포함할 경우, 소스 코드의 크기가 너무 커져 압축을 시도하더라도 커널 이미지의 용량을 초과할 수 있다. 또한, 커널 빌드 과정에서 소스 코드의 전처리 및 첨부 과정에서 여러 문제가 발생할 수 있다. 예를 들어, 'kernel-source_skin'은 전체 디렉터리를 읽는 과정에서 I/O 작업이 증가하여 대규모 시스템에서 성능 저하를 초래할 수 있다. 'kernel-source_molt'는 파일에 전체 내용이 포함되어 불필요한 데이터까지 전송하므로, 전송 데이터 양이 증가하고 이로 인해 전송 시간과 저장 공간 사용의 비효율성이 발생한다.

이러한 문제를 해결하기 위해, 수정된 부분만을 추출하여 전송하거나 더 나은 압축 알고리즘을 적용하는 방법을 통해 커널 변경 로그를 보다 효율적으로 관리하고 [그림 2]와 같이 크기를 줄일 수 있다 [8-10]. 또한, 커널 소스 코드의 디렉터리 구조를 최적화하면 읽기 및 전송 효율을 향상시킬 수 있다 [11].

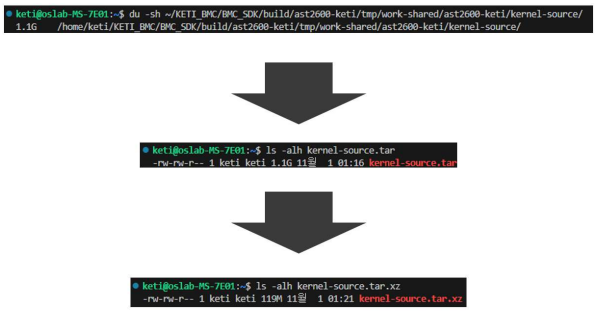

그림 2. 커널 원본의 크기와 압축 커널의 크기

2. 설계

가. 최적화된 디렉터리 구조

두 개의 커널 소스 코드 디렉터리를 비교하기 위해 전체 디렉터리 구조를 전송하지 않는다. 대신 수정된 파일명과 해당 파일이 속한 상위 디렉터리 경로와 같은 필요한 정보만을 전처리 한 후 호스트 측에 전송한다. 이러한 방법은 호스트 측에서 전송받은 정보를 효율적으로 해석하고 활용할 수 있도록 한다.

커널 소스 코드를 추가하거나 삭제한 후, 원본 커널 소스 코드에서 수정한 파일을 개별적으로 비교하는 것은 비효율적이다. 대신, 주어진 경로에 파일이 존재하는지 확인함으로써 수정된 부분을 효율적으로 찾을 수 있다. 변경된 소스 코드를 프로파일링 도구로 전송할 때 디렉터리 구조 정보를 함께 포함하면, 비용을 줄이면서 변경 사항을 효율적으로 비교할 수 있다. 이를 구현하기 위해, 커널 소스 코드의 최상위 디렉터리에서 시작하여 깊이 우선 탐색(DFS) 방식으로 파일 이름과 함께 디렉터리 구조를 트리 형태로 기록한다. 기록된 디렉터리 구조는 후처리를 위해 [그림 3]와 같은 형식으로 저장되며, 각 파일에 대해 파일 이름 길이(1바이트), 파일 이름(n바이트), 자식 파일 수(2바이트)를 포함한다. 저장된 정보를 구문 분석할 때는 첫 번째 바이트에서 파일 이름 크기를 읽고, 이어서 다음 2바이트에서 자식 파일 수를 읽어 해당 정보를 트리 형식으로 저장한다. 이후, 동일한 디렉터리 내에서 다음 파일을 읽고 기록하는 과정을 반복한다.

```
1: typedef struct _file_node {
2:     char *name;
3:     char *hash;
4:     unsigned char name_length;
5:     unsigned short child_count;
6:     struct _file_node *child_head;
7:     struct _file_node *next_node;
8: } file_node;
```

그림 3. 커널 소스의 디렉터리 구조 저장

[12]의 방법은 파일 속성을 포함한 디렉터리 구조를 압축하여 전송하는 반면, 본 논문에서 제안하는 방식은 파일 이름과 상위 디렉터리 경로만 포함하여 데이터를 단일 파일로 관리함으로써 데이터 양을 줄이고 비교 시 발생하는 I/O 비용을 감소시킨다.

나. 변경 로그 전처리

변경된 커널 소스 코드를 저장할 때 원본 커널 소스 코드와 비교하여 변경 사항만 저장함으로써 파일 크기를 줄일 수 있다. 이를 통해 저장 공간을 절약할 뿐만 아니라 수정된 부분에 집중해 데이터 관리의 효율성을 높인다.

프로파일링 도구에 수정되거나 추가된 커널 소스

코드 파일만 전송하여 네트워크를 통해 전달되는 데이터의 크기를 줄인다. 앞서 언급한 바와 같이 파일 데이터에는 경로 정보가 포함되지 않으므로 수정된 파일의 경로를 저장하는 'hash.txt'도 함께 전송된다. [그림4]와 같이 'hash.txt'의 각 줄에는 수정한 파일 이름과 해당 파일 이름을 해싱한 값을 슬래시(/)로 구분하여 저장한다.

```
32709a7e44c1e904d2bdeb04106bd40e/virt/kvm/coalesced_mmio.c
31713f6c0b9cf1098875b071777e7850/virt/kvm/coalesced_mmio.h
c3b7cf39e3af76976802dbe952c7c0b1/virt/kvm/eventfd.c
998cfcf75cbdb51de644ad7117c02240/virt/kvm/irqchip.c
e0434d1e9b05ce7829d67a5896ce0640/virt/kvm/kvm_main.c
77bda9a536b5bd61d8d6d384c6e30c6a/virt/kvm/vfio.c
4b752ee7a6058416f5af845dfc1db788/virt/kvm/vfio.h
5e0031ab6742d2cbda0a23dce0e93460/virt/lib/Kconfig
4d8141d5dda43d8bb895bcf6419f6118/virt/lib/Makefile
92123b16224ab3168678e2f69bdc0888/virt/lib/irqbypass.c
...
```

그림 4. 'hash.txt'의 파일 경로와 해시 값

[그림 5]은 수정된 파일의 변경 내역을 기록한 예시이다. 연속적으로 추가 및 삭제된 줄들을 변경 사항으로 기록하면 파일 크기를 효과적으로 줄일 수 있다. 변경 로그는 '(시작_줄_번호)-(종료_줄_번호):(추가된_줄_수)' 형식으로 작성되며, 추가된 줄에 따라 시작부터 끝까지의 줄 번호를 수정할 수 있다. 또한, 개발자는 로그의 맨 앞에 있는 추가(+), 삭제(-), 수정(*) 기호를 통해 수정된 소스 코드와 원본 소스 코드 간의 차이를 쉽게 식별할 수 있다.

추가된 줄의 경우, 원본 소스 코드에서 추가된 줄을 포함하여 '+(원본_줄_번호):(추가된_줄_수)' 형식으로 기록된다. 삭제된 줄의 경우, 시작과 종료 줄 번호를 표시하기 위해 '-' 기호를 사용하여 '-(시작_줄_번호):(종료_줄_번호)' 형식으로 기록되어 후처리 시 비교를 용이하게 한다.

```
+3:1
#include <linux/async.h>
-44:50
+58:9
static void panic_show_mem(const char *fmt, ...)
{
    va_list args;
    show_mem(0, NULL);
    va_start(args, fmt);
    panic(fmt, args);
    va_end(args);
}
*83-83:1
    panic_show_mem("can't allocate link hash entry");
```

그림 5. 수정된 파일의 변경 내역

예를 들어, [그림 5]에서 +3:1은 3번 줄에 한 줄이 추가되었음을 나타내고, -44:50은 44번 줄부터 50번 줄까지의 코드가 삭제되었음을 의미한다. 또한, 83-83:1은 83번 줄에서 수정이 발생했음을 나타내며, 원래 줄이 삭제되고 새로운 내용이 추가되었음을 보여준다.

다. 효율적인 비교 작업

호스트 측에서 소스 코드를 변경한 기록을 시각화하고 서로 다른 버전 간의 변화를 식별할 때, 비교 작업에 걸리는 시간을 최소화하는 것이 중요하다. 이를 달성하기 위해 [그림 6], [그림7]과 같이 효율적인 알고리즘과 데이터 구조를 통해 변경 사항을 빠르고 정확하게 분석한다.

가와 나에서 설명한 작업은 커널 빌드 중에 수행되며, 이 과정에서 파일이 생성 및 전송된다. 커널 부팅이 완료된 후 호스트 측에서 수정된 버전을 요청하면, 생성된 파일들이 API를 통해 전송되어 호스트 측에 저장되고 관리된다.

Algorithm 1 Initialize Directory Structure

```
function INIT_STRUCTURE(struct, count)
    i, new_struct ← 0

    for i=0 to count do
        new_struct.name_length ← read(1byte) to int
        new_struct.child_count ← read(2byte) to int
        new_struct.name ← read(name_length)

        struct.child.insert(new_struct)
    end for
end function=0
```

그림 6. 디렉터리 구조 초기화

Algorithm 2 Compare Files Between Two Version

```
function COMPARE_FILE(a, b)
    if a.path ∩ b.path = ∅ then
        return false
    end if
    if a.type ≠ b.type then
        return false
    end if
    if a.hash = b.hash then
        return true
    end if
    return false
end function
function DIFF_KERNEL_SOURCE(v1, v2)
    if compare_file(v1, v2) = true then
        return
    end if
    new_v1 ← merge(org, v1)
    new_v2 ← merge(org, v2)
    result ← LCS(new_v1, new_v2)
    print(result)
end function
```

그림 7. 두 버전의 파일 비교

파일에서 디렉터리 구조를 읽고 트리 구조를 구성하는 과정은 위의 가에서 설명한 내용의 역순이다. 첫 번째 바이트에서 파일 이름의 길이를 읽고, 다음 2바이트에서 자식 파일 수를 확인한다. 그 후, 파일 길이만큼 파일 이름을 읽는다. 위 과정을 자식 파일 수만큼 반복하여 자식 파일의 내용이 트리에 재귀적으로 삽입된다. 이 절차가 완료되면 부모 디렉터리에 상대적인 다음 파일에 대해 동일한 프로세스가 반복되어 트리 구조가 완성된다.

두 버전 간의 변경 사항을 확인하기 위해, 두 버전의 커널 소스 코드 변경 로그에 대한 후처리 정보가 트리 구조에 저장된다. 루트에서 시작하여 두 버전을 비교한다. 특정 파일의 경로가 두 데이터 세트에서 동시에 존재하지 않으면 추가되거나 삭제된 파일로 간주된다. 파일 유형이 서로 다르면(하나는 파일이고 다른 하나는 디렉터리) 해당 파일은 변경된 것으로 간주된다. 또한 해시 값이 다를 경우에도 파일이 수정된 것으로 판단한다. 변경되지 않았거나 삭제되지 않은 파일은 출력되지 않는다. 현재 변경 로그는 변경 사항만 저장하므로, 수정된 파일은 원본 버전과 비교하여 추출된다. 추출된 파일 내용은 편집거리 알고리즘(Levenshtein Distance) 알고리즘을 사용하여 차이 정보를 도출한다. 이 차이 정보는 이후 후처리 과정을 거쳐 프로파일링 도구를 사용하여 시각화된다.

3. 실험 결과

본 논문의 실험은 듀얼 코어 ARM Cortex A7 CPU가 장착된 AST2600-EVB 보드[13], DDR4-1600Mbps 이상의 속도 등급을 가진 2GB DDR4 SDRAM, SPI 플래시 메모리 저장소와 리눅스 커널 버전 5.4.62의 플랫폼 환경에서 수행한다. 제안된 비교 방법 검증을 위해, 파일 개수에 따른 방법별 디렉터리 구조 크기 비교, 파일 수정 줄 수에 따른 변경 로그 줄 수 비교가 진행되었다.

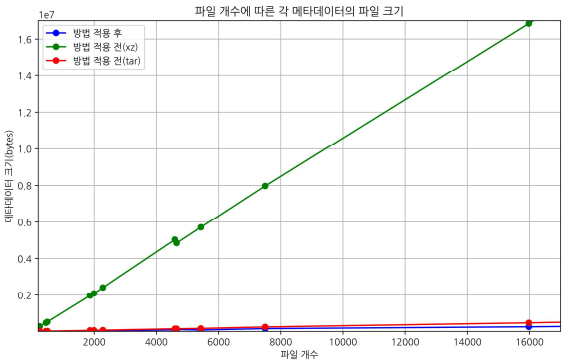

그림 8. 파일 개수에 따른
방법별 메타데이터 파일 크기

[그림 8]은 디렉터리 내 파일 개수에 따른 디렉터리 구조의 크기를 나타낸다. 모든 방법이 파일 개수에 따라 파일 크기가 선형적으로 증가한다. 결과에 따르면, 하나의 파일로 이루어진 제안된 방법의 메타데이터 파일 크기가 [그림 1]의 방법처럼 디렉터리 구조로 이루어진 메타데이터를 xz 명령어로 압축한 전통적인 방법보다 매우 작다. 디렉터리 구조의 메타데이터를 tar로 압축하는 방법과는 근소한 차이로 제안한 기법이 더 나은 성능을 보인다. 특정 구조에서는 tar 압축 방법이 더 작은 파일 크기를 가질 수 있지만, 전송 전후의 압축 및 파일 I/O에 소요 되는 시간까지 고려한다면, 제안된 방법이 tar 압축 방법에 비해 훨씬 효율적이다. 파일 I/O 시간을 줄임으로써 읽기/쓰기 속도를 증가시키기 때문에, 실시간 데이터 처리 환경이나 빈번한 파일 접근이 필요한 시스템에서 특히 더 유리하다.

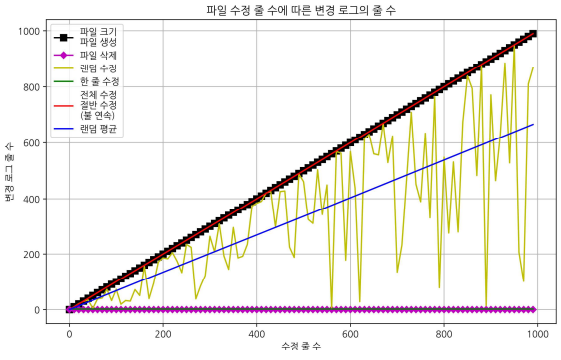

그림 9. 파일 수정한 줄 수와 변경 로그 간의 관계

파일이 한 번에 한 줄 수정될 경우, 변경 로그는 해당 파일의 연산 정보와 함께 단일 변경 로그를 기록한다. 파일이 한 번에 여러 줄 수정 되는 경우 한 줄의 연산 정보와 함께 수정의 경우 수정된 모든 줄이 로그에 기록되며 삭제의 경우 별도의 항목이 기록되지 않는다. 수정 없이 추가된다면 한 줄의 연산 정보와 추가된 항목만이 로그에 기록된다. [그림 9]는 수정된 파일의 줄 수를 10줄씩 증가하며 수정된 줄 수와 변경 로그의 줄 수와의 관계를 확인한다. 파일이 새로 생성된 경우 전통적인 방법과 제안된 방법 모두 파일 전체를 보낸다. 전통적인 방법은 파일 수정시 수정된 파일 전체를 보내는 방식을 가지고 있다. 반면에, 제안된 방법은 불연속적으로 한 줄 수정될 때마다 로그가 두 줄씩 증가한다. 불연속적인 수정은 수정된 파일의 전체 줄 수의 절반까지 증가한다. 수정된 줄 수의 총 개수가 절반 이상이라면 필연적으로 연속된 줄이 생기며, 연속한 줄의 개수가 줄어들게 된다. 이는 수정된 줄 수가 하나 증가할 때마다 변경 로그가 한 줄 감소하게 되어, 결과적으로 아무리 변경 로그의 줄 수가 많아져도 수정된 파일의 전체 줄 수 +1을 넘지 않는다. 수정된 줄 수가 파일 전체라면 파일 전체 줄 수에 연산 정보 한 줄이 증가한다.

0부터 수정된 파일의 전체 줄 수에서 무작위로 수 하나를 뽑아 수정될 라인의 수라고 가정하고, 이후 뽑은 수만큼 수정된 파일 줄 번호를 중복을 허용하지 않도록 무작위로 뽑으면 가상의 수정을 시뮬레이션할 수 있다. 이 시뮬레이션을 통해 수정된 파일의 줄 수에 따른 변경 로그의 줄 수를 구한 결과가 [그림 9]의 랜덤 수정이 된다. 한 번의 랜덤 수정에서 방법간 줄 수 비율의 평균을 구하면 평균 줄 수 비율을 구할 수 있다. 검사의 정확도를 높이기 위해 랜덤 수정을 100번 시행하여 각각의 평균 줄 수 비율의 평균을 구한 값이 [그림 9]의 랜덤 평균이다. 실험 결과 0.67정도의 줄 수 비율을 보이며, 커널 수정이 균일한 무작위 분포에서 이루어진다고 가정할 때 33%의 파일 크기 감소율을 기대할 수 있다. 게다가 변경 로그로 기록되는 줄의 대부분은 매우 작은 크기이며, 커널 수정 특성상 코드의 대규모 수정이 일어나는 경우는 드물기 때문에 실사용에서는 33%보다 더 좋은 감소율을 보일 것으로 전망한다.

III. 결 론

본 논문은 풀스택 프로파일링 도구에 특화된 커널 소스 코드 변경 로그의 효율적인 관리 및 전송을 위한 새로운 방법론을 제시한다. 파일 전체의 소스코드를 전송하거나, 모든 파일의 변경을 추적하는 전통적인 접근 방식은 많은 자원 소모와 시간을 필요로 한다. 이에 반해 본 논문의 방법은 커널의 디렉터리 구조와 해시 값을 보다 효율적인 방식으로 저장하여 빠른 추적과 비교가 가능하다.

커널 변경 로그 전송의 고유한 한계를 해결하기 위해, 디렉터리 구조 최적화, 변경 로그 전처리, 비교 오버헤드 최소화 등 여러 주요 설계 목표를 설정하였고, 이러한 목표를 준수함으로써 시스템 성능에서 상당한 최적화를 입증하였다. 수정된 파일의 경로와 해시 값에 집중함으로써 불필요한 데이터 전송을 최소화하여 네트워크 자원과 시스템 효율성을 보존하였다.

자동화된 분석 도구와 실시간 데이터 전송을 통한 버전 관리의 통합은 개발자의 작업 부담을 줄이고 시스템의 일관성을 보장한다. 이러한 접근 방식은 대규모 프로젝트에서 특히 유리하며, 체계적이고 효율적인 버전 관리를 돕는다. 또한, 확장 가능한 아키텍처와 사용자 친화적인 인터페이스는 개발자가 변경 로그를 쉽게 해석하고 분석할 수 있도록 지원한다.

향후 연구에서는 시스템 성능 및 안정성을 더욱 향상하기 위해 이 방법론을 확장할 계획이다. 대규모 클라우드 데이터 센터와 임베디드 시스템 등 다양한 환경에서 접근 방식의 적용 가능성을 평가하고, 여러 커널 버전이 시스템 성능에 미치는 영향을 검토할 것이다. 이러한 노력은 시스템 신뢰성을 개선하고 유지 관리 작업의 효율성을 높이는 한편, 개발자에게 복잡한 내부 작동에 대한 더 깊은 통찰력을 제공하여 궁극적으로 안정적이고 효율적인 시스템 기능을 보장할 것이다.

REFERENCES

[1] Technavio. *Data Center Storage Market Analysis North America, Europe, APAC, South America, Middle East and Africa - US, UK, France, China, Singapore - Size and Forecast* 2024-2028. https://www.technavio.com/report/data-center-storage-market-size-industry-analysis

[2] W. Peter. *Protecting Data Without Blowing the Budget, Part 1: Onsite Backup.* Forbes(2018). www.forbes.com/sites/forbestechcouncil/2018/10/04/protecting-data-without-blowing-the-budget-part-1-onsite-backup (accessed Nov., 2, 2024).

[3] Jessie Frazelle. 2020. "*Opening up the baseboard management controller,*" Commun. ACM, vol. 63, no. 2, pp. 38 - 40, Feb. 2020.

[4] Managing BMC-based systems(2021). https://www.ibm.com/docs/en/power9?topic=interfaces-managing-bmc-based-systems (accessed Nov., 2, 2024).

[5] 김미선, 박용석, 서재현, *"클라우드 환경에서 블록체인을 이용한 포그 기반 IoT 서비스 상호운용 시스템,"* *스마트미디어저널*, 제11권, 제3호, 39-53쪽, 2022년 04월

[6] 김재섭, 박민호, 홍지만, *"베이스보드 매니지먼트 컨트롤러를 위한 부팅 과정 프로파일링 도구,"* *스마트미디어저널*, 제11권, 제11호, 84-91쪽, 2022년 12월

[7] D. Hwang et al. 2023. "Reducing the Booting Time for BMC," RACS '23, no. 13, pp. 1 - 3.

[8] 김용민, 김희진, 김영관, 홍지만, *"하둡 클러스터의 대역폭을 고려한 압축 데이터 전송 및 저장 기법,"* *스마트미디어저널*, 제8권, 제4호, 46-52쪽, 2019년 12월

[9] Tracey L. Weissgerber et al. "From Static to Interactive: Transforming Data Visualization to Improve Transparency," *PLoS Biology,* vol. 14, no. 6, Jun. 2016.

[10] Data Compression Explained(2013). https://mattmahoney.net/dc/dce.html(accessed Nov., 2, 2024).

[11] J. Zhang et al. "Energy-Efficient Data Transmission for Underwater Wireless Sensor Networks: A Novel Hierarchical Underwater Wireless Sensor Transmission Framework," *Sensors 2023,* vol. 23, no. 12, Jun. 2023, 5759.

[12] J. Kim et al. 2023. "An Efficient Profiling Tool for Baseboard Management Controllers," *SAC '23,* pp. 1318 - 1324, 2023.

[13] AST2600(2019). https://www.aspeedtech.com/server_ast2600 (accessed Nov., 2, 2024).

저 자 소 개

오경은(준회원)

2023년 숭실대학교 컴퓨터학부 학사 졸업.
2024년~현재 숭실대학교 컴퓨터학과 석사 재학.

<주관심분야 : 운영체제, 임베디드 시스템>

김한결(준회원)

2019년~현재 숭실대학교 컴퓨터학부 학사 재학.

<주관심분야 : 운영체제, 임베디드 시스템>

오정환(준회원)

2019년~현재 숭실대학교 컴퓨터학부 학사 재학.

<주관심분야 : 운영체제, 임베디드 시스템>

안성모(준회원)

2022년~현재 숭실대학교 컴퓨터학부 학사 재학.

<주관심분야 : 운영체제, 임베디드 시스템>

홍지만(종신회원)

2003년 서울대학교 컴퓨터공학과 박사 졸업.
2004~2007년 광운대학교 컴퓨터공학과 교수.
2007년~현재 숭실대학교 컴퓨터학부 교수.

<주관심분야 : 운영체제, 임베디드 시스템>