경량 동적 코드 변환을 이용한 커널 수준 소프트웨어 계측에 관한 연구☆

Kernel-level Software instrumentation via Light-weight Dynamic Binary Translation

이 동 우* 김 지 홍** 엄 영 익***
Dongwoo Lee Jeehong Kim Young Ik Eom

요 약

코드변환 기법은 특정 명령어 집합 구조에서 작성된 프로그램 코드를 다른 구조에서 실행할 수 있도록 변환하는 일종의 에뮬레이션 기법이다. 이 기법은 주로 구형 시스템에서 동작하는 응용프로그램을 새로운 시스템에서 동작시키기 위해 사용되었다. 코드를 변환하는 과정에서 동적으로 코드를 삽입하는 것이 가능하기 때문에 소스코드의 수정 없이 기존 응용프로그램을 계측할 수 있다. 이미 응용프로그램 분야에서는 동적코드분석과 가상머신에서 이러한 기법이 널리 활용되고 있다. 반면에 운영체제의 커널은 일반적인 유저 수준의 응용프로그램과는 다른 특성을 지니기 때문에 커널 수준에서 이러한 코드변환 기법을 사용하려면 시스템 성능, 메모리 관리, 특권 명령어 처리 및 동기화와 관련된 문제가 다루어져야 한다. 본 논문에서는 커널 수준의 동적코드변환 기법을 설계하고 코드삽입을 통한 소프트웨어 계측을 제안한다. 제안기법을 리눅스 커널에 적용하여 실험을 수행하고 그 결과를 통해 본 제안기법이 커널수준에서 소프트웨어 계측에 적은 성능 부하만을 야기함을 확인하였다.

ABSTRACT

Binary translation is a kind of the emulation method which converts a binary code compiled on the particular instruction set architecture to the new binary code that can be run on another one. It has been mostly used for migrating legacy systems to new architecture. In recent, binary translation is used for instrumenting programs without modifying source code, because it enables inserting additional codes dynamically, For general application, there already exists some instrumentation software using binary translation, such as dynamic binary analyzers and virtual machine monitors. On the other hand, in order to be benefited from binary translation in kernel-level, a few issues, which include system performance, memory management, privileged instructions, and synchronization, should be treated. These matters are derived from the structure of the kernel, and the difference between the kernel and user-level application. In this paper, we present a scheme to apply binary translation and dynamic instrumentation on kernel. We implement it on Linux kernel and demonstrate that kernel-level binary translation adds an insignificant overhead to performance of the system.

☞ keyword : Binary Translation, Instrumentation, 동적코드변환, 소프트웨어 계측, 커널

1. 서 론

코드변환 기법(binary translation)은 특정 명령어 집합 구조(instruction set architecture, ISA)에서 동작하는 코드를 다른 명령어 집합 구조에서 실

* 준 회 원 : 성균관대학교 휴대폰학과 석사과정
cowsboys@ece.skku.ac.kr
** 준 회 원 : 성균관대학교 휴대폰학과 박사과정
jjilong@ece.skku.ac.kr
*** 정 회 원 : 성균관대학교 정보통신공학부 교수
yieom@ece.skku.ac.kr
[2011/04/15 투고 - 2011/05/02 심사 - 2011/07/15 심완료]
☆ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단-차세대정보컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. 2011-0020520)

☆ A preliminary version of this paper appeared in ICONI/APIC-IST 2010, Dec 16-20, Mactan Island, Philippines.This version is improved considerably from the previous version by including new results and features.

행 가능하도록 변환하는 일종의 에뮬레이션[1] 기법이다. 이 기법은 기존의 시스템에서 사용하던 응용프로그램을 새로운 시스템으로 손쉽게 이식(migration)하기[2] 위에 처음 사용되었다.

이후에 코드변환 기법은 서로 다른 명령어 집합 구조간의 시스템 이식뿐만 아니라 같은 명령어 구조 집합에서도 널리 사용되고 있다. 코드변환 기법을 적용할 때 기존의 프로그램 코드를 변환하거나 새로운 코드를 삽입[3] 할 수 있다. 이러한 점은 소스코드 수정과 같은 많은 노력을 들이지 않고도 응용프로그램을 계측할 수 있도록 한다. 프로그램 수행 시(just in-time)에 코드 변환하는 기술을 동적 코드변환(dynamic binary translation)이라고 하며 이러한 기법을 이용한 응용 분야를 동적 코드 계측(dynamic binary 계측, DBI)이라 한다.

사용자 응용프로그램 분야에서는 동적 코드 분석기와 가상 머신 모니터와 같은 시스템이 있다. Valgrind[4]와 Pin[5], Dynamo[6] 등의 동적 코드 분석기는 코드 변환 기법을 사용하여 기존 응용프로그램에 분석 관련 코드를 삽입한다. 이를 통해 기존 프로그램의 소스코드를 수정하지 않고도 머신 코드 수준의 분석이 가능하다. 또한 VMware[7], QEMU[8], Xen[9]과 같은 가상 머신 모니터 제품도 동적 코드 변환 기술을 사용한다. 가상화 환경에서 게스트 운영체제는 일반 응용프로그램과 같은 유저수준에서 동작하기 때문에 특권명령어를 실행 할 수 없다. 이러한 명령어를 동적 코드변환 기법을 사용하여 유저레벨에서 실행 할 수 있는 명령어로 바꾸어 실행한다.

커널 수준(kernel-level)에서도 동적 코드 계측 기법을 사용하여 장치 드라이버의 고장 분리 기술(device fault isolation)이나 소프트웨어 트랜잭션 메모리(software transactional memory) 등의 설계 및 구현이 가능하다. 동적 코드 변환 기법은 이미 사용자 수준의 응용프로그램에서는 널리 사용되고 있지만, 이를 커널 수준에 적용하는 것은 많은 주의가 필요하다. 커널 수준에서의 부하는 시스템 전체의 성능에 영향을 미치며 커널 내부의 메모리는 응용프로그램의 메모리 영역에 비해 한정적이다. 이외에도 특권 명령, 커널 수준의 호출 규약 및 커널에 의해 스케줄링 되는 일부의 커널 함수들이 고려 대상이 된다.

본 논문에서는 커널 수준에서 소프트웨어 계측 기법을 활용하기 위한 경량 동적 코드 변환기법을 설계하고 계측 코드 삽입 기법을 제안한다. 또한 본 기법을 최신의 리눅스 커널에 적용하여 간단한 계측을 통해 변환성능을 평가한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서 커널 수준에서의 동적코드변환 기법에 대하여 설명하고 3장에서 코드변환을 기반으로 하는 동적 계측에 대하여 알아본다. 4장에서는 실험을 통해 본 논문에서 제안하는 기법에 대하여 평가하고 5장에서 본 연구와 관련된 사용자 수준의 동적 코드변환 기법에 대하여 알아본다. 마지막으로 6장에서 본 논문의 결론을 서술한다.

2. 관련연구

유저 수준에서 동작하는 동적 코드변환이 응용프로그램의 분석 및 디버깅에 목적이 있는 데 비해 커널 수준의 동적 코드변환은 간단한 instrumentation을 성능 효율적으로 수행하는데 있다. 동적 코드 변환 기법은 이러한 목적에 따라 코드변환 과정을 크게 두 가지 방법으로 나눌 수 있다. 메모리 누수, 메모리 참조 오류 등 정적인 분석 방법으로는 불가능한 응용프로그램 분석을 위해 널리 사용되는 Valgrind, Pin, DynamoRIO[10]와 같은 동적 변환 도구는 장치 독립적인 중간언어(intermediate representation)을 사용하여 다양한 플랫폼 지원 및 자세한 분석 정보 제공 그리고 풍부한 코드 최적화를 가능 하게한다. 그러나 이 같은 접근은 매우 다양한 분석을 가능하게 하지만 변환 과정에서의 부하가 매우 크다. 이와는 달리 HDTrans, MyBT는 정보의 자세함이나 코드의 최적화 보다는 변환자체의 비용을 최소화 하는데 중점을 둔다. 테이블 기반의 변환을 통해

중간언어를 통하지 않고 직접 코드가 변환 되므로 변환 부하가 매우적다. 이러한 장점에 기인하여 가상머신[11]이나 장치 드라이버의 고장분리기법[12]등의 분야에서 연구되고 있다.

3. 커널 수준 동적코드변환

커널 수준 동적 코드 변환기는 유저 수준의 그것과 구조적으로 유사하다. 그러나 앞서 기술한 바와 같이 성능, 메모리, 특권명령, 함수 호출 처리와 관련된 몇 가지 논의점이 있다. 본 장에서는 이러한 논의들에 대해 서술한다.

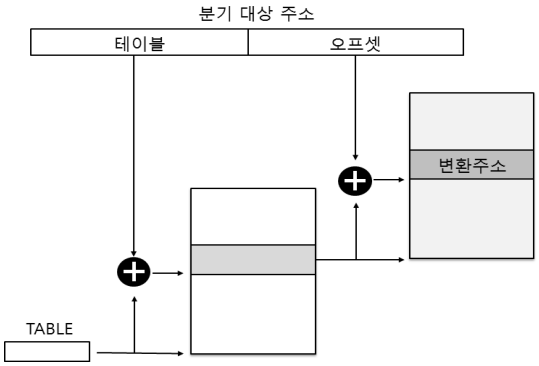
3.1 주소변환

앞에서 살펴본바와 같이 우리가 소프트웨어 계측 도구를 이용해 얻을 수 있는 장점은 매우 많다. 그러나 만약 코드를 변환하는데 드는 비용이 매우 높아 실제 실행 환경에서 사용할 수 없을 만큼의 부하가 생긴다면 이러한 기법은 무용지물이 될 것이다. 더욱이 커널 수준에서 동적 코드변환 기법의 주요 타깃은 시스템 콜 처리기(handler)나 디바이스 드라이버와 같이 매우 자주 사용되는 부분이다. 이는 커널 수준에 동적 코드 변환 기법을 적용하는 경우 변환 자체의 성능이 시스템 전체의 성능에 영향을 줄 수 있음을 의미한다.

동적 코드변환 과정에서 가장 부하가 큰 과정은 분기 명령의 대상 주소를 계산하는 것이다. HDTrans[13]의 경우 이 과정을 빠르게 처리하기 위해 해시 테이블(hash table)을 이용한다.

$$T_{addr} = H(O_{addr})$$

해시의 주소 변환 과정은 위와 같이 한 번의 해시 함수 호출로 이루어지기 때문에 시간 복잡도 O(1)의 매우 빠른 성능을 보여준다. 그러나 해시 테이블에서 충돌(collision)이 일어날 경우 이를 처리하기 위한 루틴으로 인하여 주소 계산 성능이 급격히 떨어지는 문제가 있다.



(그림 1) 분기주소 변환 과정

본 논문에서는 (그림 1)과 같이 일반적인 페이지 테이블과 유사한 래딕스 트리(radix tree)를 사용하여 분기의 대상 주소를 계산한다. 그림 1과 같이 변환 전의 대상 주소를 이등분하여 각각의 값을 각 변환 단계의 테이블을 색인(indexing)하는데 사용한다. 래딕스 트리를 이용한 주소 변환 과정은 다음과 같은 두 번의 메모리 접근 과정을 통해 수행된다.

$$T_{addr} = ref(ref(table + O_{high}) + O_{low})$$

이 방법은 O(1)의 시간 복잡도를 유지하면서 Hash에서 발생할 수 있는 충돌 문제를 해결 할 수 있다.

3.2 메모리 관리

일반적으로 동적 코드 변환기는 메모리에서 본래의 프로그램영역 외에 변환된 코드를 쓰기위한 추가적인 영역을 추가로 할당하며 이러한 영역을 코드캐시(code cache)라고 한다. 코드 변환과정에서 필요한 코드캐시 사이즈를 정확히 예측하는 것이 어려울 뿐 아니라 분기 명령으로 인한 코드 길이의 변화가 있을 수 있기 때문에 유저 수준 코드변환기는 코드캐시 영역을 충분히 크게 할당한다. 유저 수준의 응용 프로그램들은 각각 독립

된 가상 주소 공간을 사용하기 때문에 이것이 문제가 되지 않는다. 그러나 커널 수준에서는 모든 응용 프로그램들이 커널 영역 메모리를 공유하기 때문에 코드변환을 위한 코드캐시를 무분별하게 많이 할당하면 다른 커널 구성요소에서 커널 메모리 할당을 실패하게 되고 이는 시스템에 치명적인 오류를 발생시킬 수 있다. 이러한 메모리 부족 문제를 방지하기 위해서는 적절한 메모리 관리기법이 필요하다.

본 논문에서는 가비지 컬렉션 기법을 이용하여 메모리 관리를 수행한다. 동적 코드 변환 기법에 의해 변환되는 코드는 자주 실행되는 지역성(locality)이 높은 코드뿐만 아니라 오직 한번만 실행되는 코드 영역도 포함된다. 이러한 코드 때문에 해당 프로그램이 수행되는 과정에서 코드캐시가 점점 가득 차게 된다. 변환된 코드가 할당된 코드캐시 영역을 넘어 사용되면 시스템에 심각한 오류를 발생 시킬 수 있으므로 그 전에 가비지 컬렉션을 수행하여 코드 캐시 영역의 공간의 가용 영역을 늘려 준다.

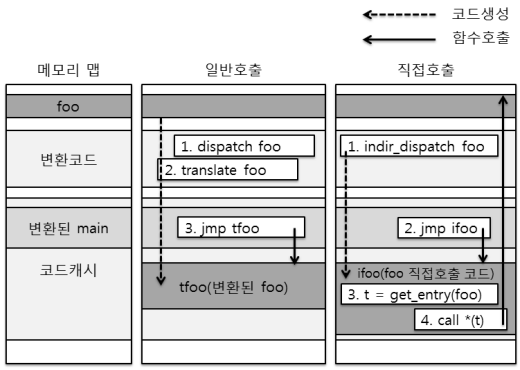
최초에 얼마나 많은 코드 캐시영역을 할당하고 언제 가비지 컬렉션을 수행할 것인지는 정책적으로 결정할 수 있는 요소이다. 코드 캐시를 크게 할당하면 가비지 컬렉션 횟수가 줄어 그에 따른 성능 부하는 줄어들지만 더 많은 메모리 영역을 사용하게 된다. 또한 가비지 컬렉션이 이루어지는 시점을 어떻게 결정하느냐에 따라 코드캐시의 가용성과 성능이 서로 상충된다. 코드 캐시 내부의 기본 블록(basic block)들의 실행 횟수에 따라 LRU (least recently used)정책을 적용하여 컬렉션 대상을 결정하는 것도 가능하다. 그러나 본 제안 기법은 HDTrans[14]와 같이 코드 캐시를 완전히 비우고 다시 코드 변환을 진행 하는 가장 기초적인 가비지 컬렉션을 적용하였다. 이러한 기법은 가비지 컬렉션 부하의 상한을 나타낸다. 4.2절에서 이러한 기법의 성능 부하를 자세히 알아본다.

3.3 커널함수 직접호출

커널의 메시지 출력을 위해 많이 사용되는 printk 커널 함수는 응용 프로그램의 printf함수와 이름은 비슷하지만 그 동작 방식에 있어서는 큰 차이를 보인다. 앞에서 살펴본 바와 같이 유저 수준에서는 특권명령을 실행 할 수 없기 때문에 응용프로그램에서 호출 하는 printf함수는 시스템 자원에 접근하기 위해 시스템 호출 명령을 사용 한다. 그러나 printk는 커널 내부의 함수이기 때문에 함수 내부에서 시스템 자원에 직접 접근하게 된다. 시스템 콘솔은 공유가능한 자원이 아니기 때문에 데이터를 출력하기 위해서는 콘솔에 대한 독점적 지위를 획득해야 한다. 이를 위해 printk는 콘솔에 대하여 잠금을 획득하고 데이터를 기록한다. 잠금을 획득한 상태에서 커널이 선점(preemption)되어 다시 printk가 호출 될 경우 커널이 데드락에 빠질 위험이 있으므로 이 경우를 대비해 커널을 선점 불가능 상태로 만들기도 한다. printk함수처럼 콘솔과 같은 공유자원을 접근하고 잠금을 획득하는 등의 동작을 하는 커널 내부 함수들이 많기 때문에 이러한 함수까지 코드 변환을 하는 것은 결과를 예측하기 어렵다.

본 제안 기법은 이러한 문제를 원천적으로 해결 할 수 있도록 함수 직접 호출 기법을 사용한다. 코드를 변환하지 않고 직접 호출하기 때문에 변환에 따른 부하를 줄일 수 있으며 커널 내부 함수의 일관성을 유지할 수 있다. 그러나 함수 호출 과정은 상대주소 호출 방식(relative addressing mode)을 사용하기 때문에 변환된 코드에서 커널 함수를 직접 호출하기 위해서는 특별한 처리가 필요하다. 이러한 처리를 위해 MyBT[15]에서 사용된 경량 동적코드 변환 기법을 사용한다.

(그림 2)는 본 동적코드변환 과정에서 사용되는 함수 호출 방법과 MyBT에서 사용된 함수 호출 과정을 비교한 그림이다. 일반호출 과정에서는 코드를 변환하고 변환된 함수를 호출하지만, 직접 호출 기법은 코드를 변환하는 과정이 없다. 대신 상대주소 호출 방식을 대체하기 위해 간접주소



(그림 2) 함수 호출 방법 비교

호출 방식(indirect addressing mode)을 사용한다. 대상 함수의 주소를 직접호출코드 생성 시 동적으로 결정(resolving)하여 간접주소 호출방식을 통해 직접 대상 함수를 호출 한다.

4. 커널 계측

3장에서는 커널 수준에서의 계측을 위한 동적 코드변환 기법에 대하여 제안하였다. 본 장에서는 실제로 코드 계측을 위한 코드 삽입 기법을 설명한다. 코드 변환과정에서 코드의 무결성(integrity)을 해치지 않으면서 필요한 코드를 적절한 위치에 삽입하고 해당 코드를 최적화 하는 기법을 제안한다.

4.1 코드삽입

동적 코드 변환과정은 다음과 같다. 1)프로그램을 각 명령 줄(instruction) 별로 해당 명령어 집합 구조의 디코딩(decoding)규칙에 따라 디코딩 한 후, 2)해당 명령줄의 명령(opcode)과 피연산자(operand)를 파싱(parsing)한다. 3)해당 정보를 기반으로 코드를 변환하여 코드캐시에 쓰고(emitting) 4) 실행흐름을 프로그램으로 넘겨 코드캐시의 코드를 실행한다(execution). 이와 같이 코드 변환 과정에서 해당 명령줄의 명령과 피연산자 정보를 얻을 수 있기 때문에 이를 기반으로 계측 코드를 삽입 할 수 있다.

프로그램을 계측하기 위해 일반적으로 명령어, 기본 블록, 함수 혹은 프로시저, 프로그램 단위의 시작 지점과 끝 지점 두 곳이 코드 삽입의 대상이 된다. 본 기법의 활용도를 높이기 위해 다수의 계측이 가능해야 한다. 다음과 같은 코드 조각을 예를 들어본다.

```
…
mov     (%ebx), %eax
idiv  (%ecx)
…
```

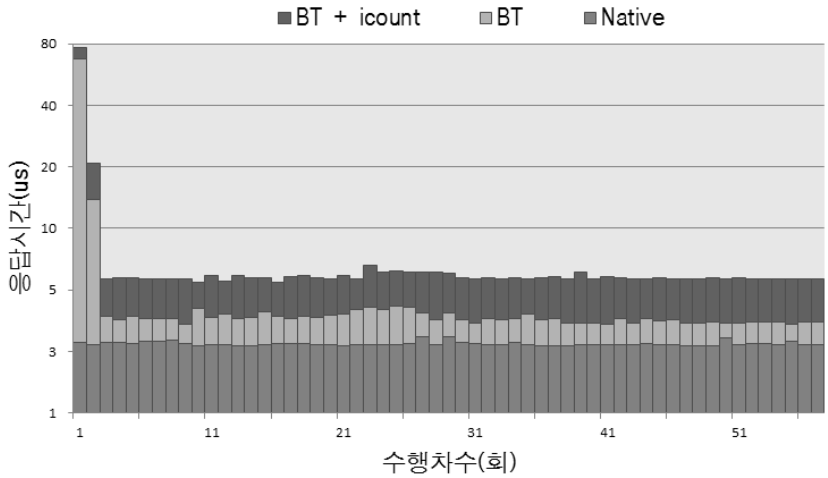
만약 이 코드 조각에 대해 0으로 나누기 오류(devide by zero fault) 방지와 널(null) 포인터 참조 오류(null pointer reference fault) 방지를 위한 계측을 함께 적용한다면 다음과 같은 형태가 된다.

```
        …
        cmp     %ebx, 0
        je      <error_ref_null>
        mov     (%ebx), %eax
        cmp     %ecx, 0
        je      <error_ref_null>
        cmp     (%ecx), 0
        je      <error_div_by_zero>
        idiv    (%ecx)
        …
<error_div_by_zero:>
        …
<error_ref_null:>
        …
```

mov 명령의 경우 단순히 널 포인터 참조에 대한 에러만을 체크하면 되지만, idiv 명령의 경우 메모리 참조를 통해 나누기를 처리하기 때문에 두 가지 계측을 차례로 사용한다.

4.2 코드 최적화

계측 코드의 삽입은 일반적으로 함수 호출의 형태로 이루어진다. 그러나 함수 호출의 과정은 함수 인자 전달을 위한 동작을 하고 반환 주소를 스택에 넣은 후 새로운 스택 프레임을 만들고 함수가 실행되며 함수가 종료된 이후에 다시 스택



(그림 3) 동적 코드변환 매 수행 별 응답시간

프레임을 복원하는 등 복잡한 과정을 거치게 된다. 더욱이 계측 코드의 경우 대상 프로그램의 무결성을 해치지 않도록 원래 프로그램의 상태(state)를 저장/복구하는 과정도 필요하다. 명령어 단위의 계측을 수행하는 경우 매 명령마다 위와 같은 과정을 거치게 되므로 전체 프로그램 성능에 큰 부하를 야기한다.

이러한 문제점을 해결하기 위해 코드의 진입(entry) 및 진출(exit)의 부하를 줄여야 한다. 이를 위해 프로그램단위 혹은 기본 블럭 단위의 레지스터 재 할당[16](register reallocation)과 같은 기법을 적용 할 수 있다. 본 논문에서는 구현의 편의성을 위해 이러한 기법 적용은 향후 연구로 남겨둔다. 그러나 실험을 통해 최적화 하지 않은 코드 삽입의 성능을 측정하여 코드 삽입에 따른 부하의 상한을 알아본다.

5. 실험평가

이 장에서는 본 논문의 제안 기법을 리눅스 커널에 적용한 후 간단한 문자 장치 드라이버를 동적으로 변환하여 수행하는 실험을 통해 성능을 평가한다. 실험은 Intel(R) Core(TM) i5-430M @ 1.98GHz, 4GB RAM의 하드웨어 시스템 상에서 Ubuntu 10.04 커널버전 2.6.38.2의 환경에서 수행하였다.

간단한 문자 장치 드라이버에 본 제안기법을 적용한 후, 응용프로그램에서 해당 드라이버에 I/O 동작을 요청하고 그 응답시간을 측정하는 실험하였다. (그림 3)은 동적 코드변환을 적용하지 않았을 때(Native), 동적 코드변환만 적용했을 때(BT), 동적 코드변환을 통해 수행하는 총 명령줄 수를 세는 간단한 계측을 수행했을 때(BT + icount)의 성능을 비교한 결과이다. 원래의 드라이버의 경우 자신의 코드를 실행하는 것 외에 추가적인 부하가 발생하지 않으므로 처음부터 일정한 성능을 보여준다. 코드변환을 적용한 경우는 처음 수행 시에는 응답시간이 매우 크지만 수행 횟수가 어느 이상이 되면 매우 적은 부하만을 나타내며 수행되는 것을 확인 할 수 있다. 이는 해당 코드가 처음 실행될 때는 코드 변환과정이 필요하기 때문에 이로 인한 성능 부하가 나타나지만 모든 코드에 대한 변환이 끝난 이후에는 원래의 드라이버와 마찬가지로 자신의 코드캐시 영역에서 코드를 실행하기 때문이다. 명령 줄 수를 세는 계측을 적용한 경우도 코드 변환을 적용했을 때와

(표 4) 코드변환 응답시간 평균

항목	시간	비율
Native	3.22us	-
BT	3.58us	10.05%
BT + icount	6.14us	90.61%

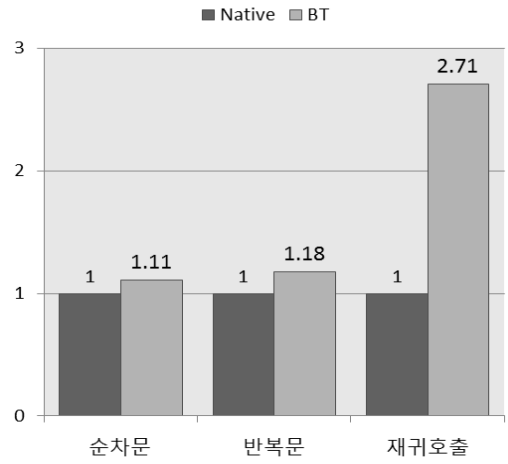
(표 5) 함수 직접호출 적용 시 응답시간 평균

항목	시간	비율
BT	3.83us	-
BT+direct	3.63us	-4.76%

마찬가지의 부하 양상을 확인 할 수 있다. 다만 계측에 의한 추가적인 부하로 인해 평균적인 응답시간이 더 늘어 난 것을 확인 할 수 있다.

(표 4)는 각 경우의 평균적인 성능 부하 량을 나타내고 있다. 코드 변환 시 발생하는 약 10%의 부하는 분기 명령의 처리를 위해 추가된 코드로 인한 것이다. 이러한 성능부하는 각 Basic Block의 코드길이에 따라 달라질 수 있다. 계측에 의한 약 90%의 부하는 매 명령 줄 마다 계측이 수행되어 본래의 드라이버 코드보다 더 많은 수의 명령이 실행되었기 때문이다. 이러한 추가적인 80.56%의 부하는 계측을 수행하기 위한 최소의 부하를 나타낸다. 그러나 이는 3.2절에서 설명한 바와 같이 계측 코드 삽입 과정에서 어떠한 최적화도 수행하지 않은 결과이므로 향후 연구에서 코드 삽입과정에 최적화를 수행하고 특정 명령줄에만 적용되는 계측의 경우 이보다 낮은 부하를 나타낼 것이다.

(표 5)는 코드 직접 호출 기법의 성능을 실험한 결과이다. 코드 변환 대상이 되는 장치 드라이버에서 사용되는 모든 커널 함수를 변환 하여 호출 하였을 때(BT)와 변환하지 않고 직접 호출 하였을 때(BT+direct)를 나타낸다. 2.3절에서 설명한 바와 같이 직접 호출을 통해 호출 되는 함수의 경우 변환과정의 부하가 없고, 함수 코드를 직접 실행하기 때문에 분기 명령 등의 처리로 인한 부

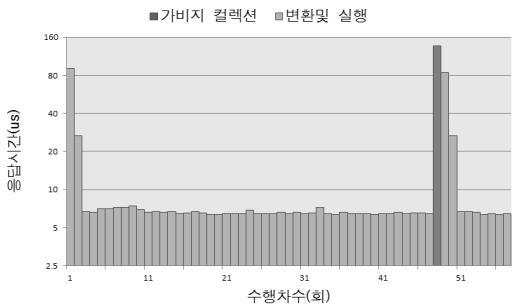


(그림 4) 분기 명령별 성능 부하 비교

하도 발생하지 않는다. 단지 간접 주소 호출 방식으로 인한 분기 대상 주소 변환과정으로 인해 성능의 이점은 대략 5%정도이다. 그러나 앞서 설명한 바와 같이 함수 직접 호출 기법은 커널 내부의 동기화로 인한 문제를 해결하기 위한 기법으로 더 큰 의미를 갖는다.

(그림 4)는 분기 명령의 성능 부하를 더 자세히 알아보기 위해 순차 문, 반복 문, 재귀 호출을 이용하여 성능 실험을 한 결과이다. 순차문의 경우 앞서 수행한 실험과 같이 대략 11%가량의 성능 부하를 보이는 반면 반복문의 경우 7%의 부하가 추가적으로 발생한다. 반복적인 함수 호출로 수행되는 재귀호출의 경우 200%에 이르는 부하를 보여준다. 이는 본 제안 기법의 함수 호출 부하가 매우 크다는 것을 알 수 있다. 이는 향후 연구에서 큰 개선의 여지가 있음을 말해준다.

(그림 5)는 가비지 컬렉션 성능을 실험한 결과이다. 각 수행차수에서 그림3과 유사한 실행 양상을 보이지만 계속된 코드 변환 및 실행으로 코드 캐시가 가득 차게 되면 인해 가비지 컬렉션이 이루어 졌다. 본 실험 결과에서 나타난 가비지 컬렉션의 부하는 코드캐시의 크기에 관계없이 항상 일정하기 때문에 일반적으로 사용되는 장치 드라이버의 경우 코드변환에 의한 부하에 비해 이러



(그림 5) 가비지 컬렉션 성능 실험 결과

한 가비지 컬렉션의 부하는 매우 적은 부분만을 차지한다.

6. 결 론

본 논문은 경량 동적코드변환 기법을 이용한 커널 수준에서의 소프트웨어 계측 기법 및 구현에 대하여 제안 하였다. 리눅스 운영체제 커널에 제안기법을 적용하여 성능 실험을 수행하였다. 본 제안 시스템의 주요 타깃인 운영체제의 장치 드라이버는 시스템이 시작되고 종료될 때 까지 반복적으로 사용되므로 코드가 모두 변환된 이후에는 실험에 사용된 응용 프로그램이 큰 성능 부하 없이 동작하였다. 즉 본 논문의 코드변환 및 계측 기법은 적은 부하만을 야기함을 확인 하였다.

동적코드변환은 이미 많은 연구가 진행된 분야이지만, 변환된 코드의 분기 주소를 매번 계산해 주어야 하는 부하와 가비지 컬렉션에 따른 부하 그리고 코드가 삽입된 기본 블록의 최적화 문제 등 성능향상과 메모리관리에서 더 많은 개선의 여지가 남아있다. 향후 이러한 개선점에 대한 추가 연구가 진행 되어야 할 것이다.

참 고 문 헌

[1] M. Probst, “Dynamic Binary Translation,” In Proc. Of the UKUUG Linux Developers’ Conf., Jul. 2002.

[2] M. Probst, “Fast Machine-Adaptable Dynamic binary Translation,” ACM SIGPLAN Not., Vol. 35, no. 7, pp.41-51

[3] N. Nethercote and J. Seward, “Valgrind: A Program Supervision Framework,” Electronic Notes in Theoretical Computer Science, Vol. 89, no.2, pp.44-66, 2003.

[4] N. Nethercote and J. Seward, “Valgrind: a framework for heavyweight dynamic binary instrumentation,” in Proc. Of the 2007 PLDI Conf., Vol. 42, pp.89-100, 2007.

[5] CK. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, VJ. Reddi and K. Hazzelwood, “Pin: building customized program analysis tools with dynamic 계측,” in Proc. of the 2005 ACM SIGPLAN PLDI conf., pp.190-200, 2007.

[6] V. Bala, E. Duesterwald and S. Banerjia, “Dynamo: a transparent dynamic optimization system,” ACM SIGPLAN Not., Vol. 35, no. 5, pp.1-12, 2000.

[7] O. Agesen, A. Garthwaite, J. Sheldon and P. Subrahmanyam, “The Evolution of an x86 Virtual Machine Monitor,” ACM SIGOPS Operating Systems Review, Vol. 44, no. 4, pp.3-18, Dec. 2010.

[8] F. Bellard, “QEMU, a fast and portable dynamic translator,” In Proc. Of the 3rd Ann. Haifa Experimental Systems Conf., pp.24-26, May 2010

[9] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, “Xen and the Art of Virtualization,” In Proc. Of the ACM Symp. on Operating System Principles Oct. 2003

[10] D. Bruening, T. Garnett, and S. Amarasinghe. “An infrastructure for adaptive dynamic optimization.” In Proc. Of Intl. Symp. on Code

Generation and Optimization, pp.265-276, Mar. 2003.

[11] P. P. Bungale, S. Sridhar, and J. S. Shapiro, “Supervisor-Mode Virtualization for x86 in VDebug,” Tech. Rep. SRL2004-01, Johns Hopkins University Systems Research Laboratory, May 2004.

[12] P. Feiner, A. D. Brown, and A. Goel, “Transparent Fault Isolation using Dynamic Compilation,” Poster at 15th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Mar. 2010.

[13] S. Sridhar, J. S. Shapiro and P. P. Bungale, “HDTrans: an open source, low-level dynamic 계측 system,” In Proc. of 2006 Int. Conf. on Virtual Execution Environments (VEE), pp.175-185, Ottawa, Ontario, Canada, Jun. 2006.

[14] S. Sridhar, J. S. Shapiro, P. P. Bungale, “HDTrans: a low-overhead dynamic translator,” ACM Computer Architecture News, vol.35, no.1, pp.135-140, Mar. 2007.

[15] 김지홍, 김인혁, 엄영익, “Indirection 기법을 이용한 경량 동적 코드 변환 기법,”, 정보과학회논문지 : 컴퓨팅의 실제 및 레터 Vol. 17, no. 3, pp.170-174, 2011. 03.

[16] Z. Cai, A. Liang, Z. Qi, L. Jiang, X, Li, H, Guan and Y. Chen, “Performance Comparison of Register Allocation Algorithms in Dynamic Binary Translation,” In Proc. Of Int. Conf. on Knowledge and Systems Engineering, pp.113-119, 2009.

◐ 저 자 소 개 ◑

이 동 우

2004년 성균관대학교 컴퓨터공학과(공학사)
2010년~현재 성균관대학교 휴대폰학과 석사과정
관심분야 : 가상화 기술, 시스템소프트웨어, 운영체제
E-mail : cowsboys@ece.skku.ac.kr

김 지 홍

2008년 광운대학교 전자공학과(공학사)
2010년 광운대학교 전자공학과(공학석사)
2010년~현재 성균관대학교 휴대폰학과 박사과정
관심분야 : 가상화 기술, 시스템소프트웨어, 운영체제
E-mail : cowsboys@ece.skku.ac.kr

엄 영 익

1983년 서울대학교 계산통계학과(이학사)
1985년 서울대학교 대학원 전산과학과(이학석사)
1991년 서울대학교 대학원 전산과학과(이학박사)
1993년~현재 성균관대학교 정보통신공학부 교수
관심분야 : 분산 컴퓨팅, 시스템소프트웨어, 가상화 기술, 미들웨어, 시스템 보안
E-mail : yieom@ece.skku.ac.kr