가상 메모리 압축을 위한 CAMD 알고리즘 설계

장 승 주†

요 약

본 논문에서는 가상 메모리 압축 알고리즘으로 CAMD 알고리즘을 제안한다. CAMD 알고리즘은 페이지 폴트가 일어났을 때 이들 페이지들을 스왑 디바이스로 이동시키지 않고 주기억장치 내의 압축된 캐시 영역을 할당하여 압축된 페이지를 저장한다. 이렇게 함으로써 스왑 디바이스로 이동하는 시간과 횟수를 감소시켜서 페이지 폴트 응답시간을 줄이며 주기억장치에 저장되는 페이지들의 공간 활용도를 높일 수 있다. 메모리 내의 데이터는 일반적인 압축 알고리즘에서 다루는 데이터와는 다른 특징들을 가지고 있어서 메모리 내의 주소 값이나 배열 데이터와 같은 요소들을 고려하여 압축될 때의 효율성을 높일 수 있다.

Design of the Compression Algorithm for in-Memory Data of the Virtual Memory

Seung Ju Jang†

ABSTRACT

This paper suggests the CAMD(Compression Algorithm for in-Memory Data) algorithm that is not moved the pages into the swap space by assigning the compressed cache area in the main memory. The CAMD algorithm that supports the virtual memory system takes high memory usability and performance benefit by reducing the page fault. The memory data is not general data. It is extraordinary data format. In general it consists of specific form of data. Therefore, the CAMD algorithm can compress this data efficiently.

키워드 : CAMD 압축 알고리즘, 가상 메모리 시스템(Virtual Memory System), 가상 메모리 압축(Compression) 알고리즘, CAMD 압축 알고리즘 동작

1. 서 론

운영체제에서 메모리 관리 서브시스템은 가장 중요한 부분 중 하나이며, 물리적인 메모리의 한계를 극복하기 위한 여러 기법들이 개발되었는데, 이중 가상 메모리 기법이 가장 핵심적인 기능으로 사용되고 있다. 가상 메모리 시스템에서 메모리를 페이지(page)로 쪼개고 시스템이 실행되면서 이들 페이지를 보조기억장치로 스왑(swap)한다[11]. 리눅스 시스템에서도 이와 같은 가상 메모리 시스템을 사용하고 있다. 그러나 임베디드 시스템은 기본적으로 제한된 메모리 용량으로 인하여 사용상에 상당한 제약 사항을 가지고 있다.

가상 메모리 시스템에서 페이지 폴트(fault)가 일어날 때 주기억 장치 공간에 여유가 없게되면 특정한 페이지들이 스왑 공간으로 이동을 하게 된다. 이때 페이지들이 느린 스왑 바이스로 이동하기 때문에 성능 저하가 발생된다. 페이지 폴트가 발생한 페이지를 주기억 장치로 읽어 들이기 위해서 기존 메모리를 차지하고 있는 페이지를 스왑 디바이스로 이동시킨다. 페이지를 스왑 아웃 디바이스로 이동시킬 때 페이지 교체 알고리즘을 사용한다. 페이지 교체 알고리즘으로 많이 사용하는 것이 LRU(Least Recently Used), LFU (Least Frequently Used) 알고리즘과 FIFO(First In First Out) 알고리즘 등이 있다[14, 15].

그리고 이 페이지들이 다시 필요하면 스왑 디바이스에서 주 기억장치로 읽어 들이게 된다. 이렇게 페이지 폴트가 일어났을 때, 이 페이지들을 스왑 디바이스로 이동시키는 과정에서 보조기억장치로의 입출력 연산으로 인하여 성능 저하가 발생한다.

본 논문에서 제안하는 CAMD 알고리즘은 스왑 아웃되는 페이지들을 가능하면 줄이기 위한 방안으로 고안되었다. 본 논문에서 제안하는 CAMD 알고리즘을 사용함으로써 주 기억장치 공간을 늘리는 효과를 만들 수 있다[6]. 본 논문에서는 제안하는 CAMD(Compression Algorithm for in-Memory Data) 압축 알고리즘은 사전을 기반으로 한 LZ 계열의 압축 알고리즘이다. CAMD 압축 알고리즘에서 사용한 사전의 크기는 워드 단위 16개의 버퍼를 사용한다. CAMD 압축 알고리즘은 32비트 단위로 압축을 처리하며, 페이지 크

※ 본 논문은 2002년도 동의대학교 자체 학술연구비 지원을 받아 이루어졌음.
이 논문은 2003년도 Brain Busan 21 사업에 의하여 지원되었음.
† 정 회 원 : 동의대학교 컴퓨터공학과 교수
논문접수 : 2003년 7월 22일, 심사완료 : 2004년 6월 5일

기인 4K바이트 만큼 처리를 하면 압축 과정은 종료가 된다. 먼저 사전을 초기화하고 페이지의 데이터를 32비트 단위로 처리를 한다. 그리고 32비트의 입력 값에 대한 패턴 분류를 한다. 입력값이 '0x0' 혹은 '0xffffffff'인 경우와 그렇지 않은 경우로 분리한다. 또한, 압축을 할때 사용하는 사전으로 16엔트리의 해쉬 테이블을 사용한다.

본 논문에서 제안하는 CAMD 압축 알고리즘은 적은 해쉬 공간을 이용하여 소량의 데이터를 사용하는 가상 메모리 공간 데이터를 다룬다. 모든 비트가 0 또는 모든 비트가 1인 데이터가 많이 발생하는 가상 메모리의 데이터 특성을 최대한 이용하여 최소한의 압축 데이터 길이를 갖도록 한다. 이런 데이터 이외에 일반적인 데이터들에 대해서는 적은 해쉬 테이블 엔트리(16엔트리)를 이용하여 압축 및 압축을 푸는데 따른 부담을 최소화시켰다.

본 논문에서는 페이지 폴트가 일어났을 때 이들 페이지들을 스왑 디바이스로 이동시키지 않고 주 기억장치 내의 압축된 캐시 영역을 할당하여 CAMD 압축 알고리즘을 사용하여 압축된 페이지를 저장한다. 이렇게 할 경우 스왑 디바이스로 이동하는 시간과 횟수를 감소시켜 페이지 폴트 응답시간을 줄일 수 있다. 또한, 페이지들의 공간 활용도를 높일 수 있다. 특히 적은 저장 공간을 가진 시스템에 사용하면 효과적인 주 기억장치 관리가 가능하다.

그리고 본 논문의 구성은 2장에서 관련 연구를 살펴보고, 3장에서 CAMD 압축 알고리즘의 설계, 4장에서는 CAMD 압축 알고리즘의 구현에 대해서 설명한다. 5장에서는 실험 및 성능 평가를 설명보고 마지막으로 6장에서 결론을 내리도록 한다.

2. 관련 연구

가상 메모리 압축 시스템은 1993년에 Fred Douglis에 의해 연구가 되었다. Douglis의 연구는 DECstation 5000/200 워크스테이션에 Sprite OS를 사용하여 가상 메모리 압축 시스템을 실험했다[4]. 그의 실험은 가상 메모리 시스템에서 페이지 폴트가 발생하여 스왑 아웃될 때 페이지들을 스왑 영역으로 이동시키는 과정에서의 부하를 줄이기 위한 것이며, 유동적인 크기의 원형 버퍼 구조를 가진 압축된 캐시 영역을 만들고, 이 영역에 스왑 아웃되어 스왑 디바이스로 이동하는 페이지들을 저장하여 스왑 디바이스로의 입출력 횟수를 줄임으로써 전체 시스템의 성능을 향상시켰다. 이 때 사용된 압축 알고리즘이 LZRW1이다[10].

Douglis의 연구 이후 Doug Banks와 Mark Stemm은 포터블 디바이스 내에서의 압축된 메모리 시스템을 제안했다. Douglis의 연구의 문제점으로 데스크탑이나 워크스테이션에서는 압축된 메모리 시스템에서의 시스템 성능향상은 기대할 수 없었다. 단지 포터블 디바이스에 적합한 압축 메모리 시스템을 고안하였다[1].

그러나 그들의 연구는 포터블 디바이스라는 한정된 영역에서 설계되었으며, 보조기억 장치가 로컬이 아닌 느린 통신 수단으로 연결된 디바이스에 접근하는 경우의 실험을 하였으므로, 일반적인 PC나 워크스테이션에서는 적용하기 힘들다는 문제점이 있다[1].

압축 알고리즘에서 고려해야 할 사항은 압축률과 실행 속도, 그리고 메모리의 요구량을 우선적으로 생각해야 한다[5]. 리눅스 운영체제의 경우 일반적으로 8K 바이트 페이지 크기나 4K 바이트 페이지를 사용하고 있다. 그에 비해 임베디드 리눅스의 경우는 512 바이트 혹은 1K 바이트 페이지를 사용하고 있으므로 압축 알고리즘은 범용적인 알고리즘 보다는 적은 양의 압축에서 효율이 높고 압축 알고리즘으로 동작 원리가 단순한 알고리즘을 적용하여 페이지의 스왑 아웃되는 속도와 압축하는 속도의 차이를 감소시킨다면 시스템의 성능 향상을 기대할 수 있다. 그리고 포터블 디바이스의 경우 한정된 저장 공간을 사용하고 있다. 따라서 압축된 스왑 디바이스의 사용으로 저장 공간을 늘린다면 적은 저장 공간을 응용 프로그램이나 데이터를 저장하는데 사용하여 시스템의 저장 용량을 향상시킬 수 있다.

압축 알고리즘의 종류에는 크게 무손실 데이터 압축과 손실 데이터 압축으로 나눌 수 있다. 무손실 데이터 압축의 종류에는 RLC(Running Length Coding), VLC(Variable Length Coding), Huffman coding, DBC(Dictionary Based Coding), LZ(Lempel-Ziv)-base와 같은 것들이 있다 [12]. 본 논문에서는 LZ 기반의 알고리즘인 LZW 압축 알고리즘을 기반으로 하여 수정한다. LZ 기반 알고리즘은 압축할 자료를 코드화 하면서 기억장치 내 문자열에 대한 색인표(Index Table)를 생성하여 한 문자열씩 확인하여 그 내용이 같은 문자열을 다시 만나면 그것을 미리 간직해둔 하나의 색인을 갖게 된다. 그렇게 중복되는 모든 부분들은 색인표를 가리키는 하나의 색인을 가지기 때문에 크기를 줄일 수 있다.

3. CAMD 압축 알고리즘 설계

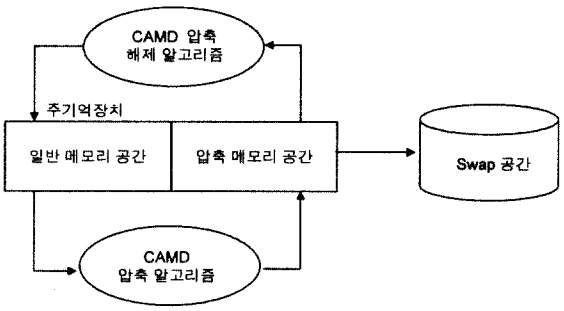
3.1 CAMD 가상 메모리 압축 시스템의 구조

리눅스의 메모리 관리 기법에서는 요구 페이징과 스왑핑 기법에 의해 페이지들이 더티 페이지(dirty page)가 되어 스와핑이 일어나면, 실행중인 프로세스는 입출력 연산이 발생하여 대기 상태로 전환 되면서 많은 시간을 기다려야 한다. 이 때 시스템의 성능 저하가 발행하는데 CAMD 가상 메모리 압축 알고리즘은 스왑 디바이스로 스와핑하는 페이지들을 주기억장치의 일부분에 압축하여 저장함으로써 스왑 디바이스로의 접근을 줄이는 방법으로 시스템의 전체적인 성능을 향상시킨다[3].

CAMD 압축 알고리즘의 동작 구조는 다음 (그림 1)과 같다.

(그림 1)에서 CAMD 압축 알고리즘을 이용한 가상 메모리 시스템의 구조를 나타낸다. 실제 메모리를 압축 메모리 영역과 비압축 메모리 영역으로 나눈다. 스왑 디바이스로 이동하는 페이지들을 압축 메모리 공간에 압축하여 저장을

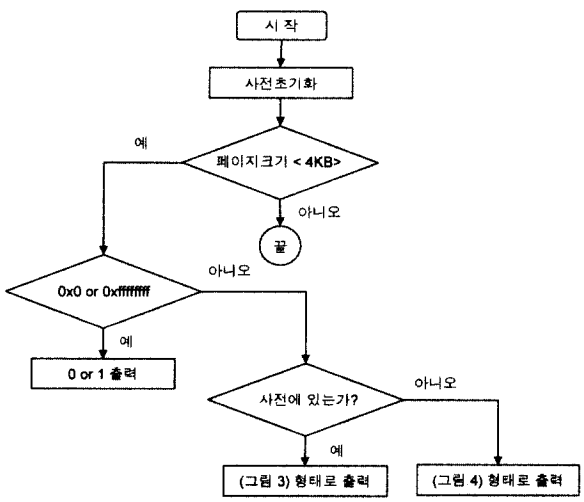
시킨다. 그리고 스왑 아웃된 페이지를 찾을 때 압축된 공간에서 페이지를 찾아서 발견이 될 경우는 압축된 페이지의 압축을 풀어서 비압축 영역으로 가져오게 된다. 이 때 압축 영역에 압축된 페이지들로 꽉 차게 되면 압축 메모리 영역 공간을 확보하기 위하여 압축 영역에 있는 페이지 데이터들을 스왑 디바이스로 압축을 풀어서 보내게 된다[2].



(그림 1) CAMD 가상 메모리 동작 구조

3.2 CAMD 압축 알고리즘

본 논문에서 설계한 CAMD 압축 알고리즘은 사전을 기반으로 한 압축 알고리즘이다. CAMD 압축 알고리즘에서는 워드 단위로 16개의 버퍼를 설정하여 사전으로 사용한다. (그림 2)는 CAMD압축 알고리즘에서 압축 과정을 플로 차트로 보여주는데, 먼저 입력 값이 단순한 값을 갖는 경우와 랜덤한 값을 갖는 경우로 나눈다. 단순한 값을 갖는 경우는 '0x0' 혹은 '0xffffffff'과 같은 경우를 생각할 수 있다. '0x0' 혹은 '0xffffffff'인 경우의 압축은 입력 값을 '0x00000001'과 마스크 연산을 하면 '0x0'인 경우는 0이 출력될 것이고 '0xffffffff'인 경우는 1 출력된다. 따라서 입력 데이터가 '0x0'인 경우는 압축된 데이터로 '00'(0x0)이 출력되고 '0xffffffff'인 경우는 '01'(0x1)이 출력된다. 그리고 압축된 데이터는 "Compressed_Buffer"에 '0x0'과 '0xffffffff'의 압축 결과인 '00'(0x00)혹은 '01'(0x01)을 나타내는 2비트만 출력된다.



(그림 2) CAMD 압축 알고리즘 동작 과정

그리고 입력값이 랜덤한 경우는 입력 값이 사전 데이터에 등록이 되어 있는지 없는지를 먼저 판별한다. 만약 입력 데이터가 사전에 등록이 되어있다면 압축 제어 비트(2비트) '10'(0x2)과 사전의 인덱스 값 4비트를 합한 6비트를 출력한다. 출력값은 '0x20'에서 2는 압축 제어 비트 '10'(0x2)에 해당하며 0은 사전 인덱스 값이 더해질 4비트의 공간을 할당한 것이다. 그리고 이 값을 사전 인덱스 값 4비트와 OR 연산을 하면 압축이 된 결과를 생성할 수 있다.

0x02	사전인덱스

(그림 3) 사전에 등록된 데이터에 대한 압축 결과

입력값이 사전에 등록된 값이 아닌 경우는 압축 제어 비트 '11'(0x3)과 입력값 전체 32비트를 합친 34비트를 압축된 출력 값으로 사용한다. 그리고 이 입력 값을 사전에 등록한다. 사전에 등록할 때는 입력 값의 마지막 4비트의 값에 따라 사전 인덱스에 해당하는 엔트리에 등록을 한다. CAMD 알고리즘에서 사전의 크기는 16개의 엔트리를 갖는다. CAMD 알고리즘에서 사전의 주소값은 입력 값의 마지막 4비트를 사용한다. 이 4비트는 CAMD 알고리즘에서 사용하는 사전의 16개 엔트리 주소를 표현할 수 있다.

출력값은 34비트로 나타내게 되는데 2번의 출력으로 압축된 데이터를 출력한다. 첫 번째 출력값은 압축 제어 비트 2비트를 출력하고, 두 번째 출력 값은 32비트 입력 값을 출력한다.

0x3	입력값 32비트

(그림 4) 사전에 등록되지 않은 데이터에 대한 압축 결과

CAMD 알고리즘의 동작 과정은 위 (그림 2)와 같다. CAMD 알고리즘은 두 가지 측면을 고려해야 한다. 하나는 압축률이고 다른 하나는 압축을 하는데 소요시간이다. 이 두 가지를 적절히 조화시키는 것이 필요하다. CAMD 알고리즘은 해시 기법을 사용하여 사전에 접근하도록 함으로써 압축을 하는데 소요시간을 줄이도록 하였다. 해시 함수는 입력값의 하위 4비트를 사전의 인덱스 값으로 사용한다. 이렇게 단순한 해쉬 함수를 사용함으로써 해쉬함수를 이용한 사전의 주소값을 계산하는 부담을 줄일 수 있다.

CAMD 압축 알고리즘에서 압축된 데이터를 푸는 과정은 압축하는 과정의 역으로 동작시키면 된다. 압축된 데이터의 형태는 아래 (그림 5)와 같으며, 압축된 데이터가 어떤 형태의 데이터(특정한 값을 갖는 경우 또는 랜덤한 값을 갖는 경우)인지 나타내는 제어비트 2비트와 제어비트에 따라 결정이 되는 데이터 비트로 구성된다.

압축 제어 비트	데이터

(그림 5) 일반적인 CAMD 압축 데이터의 생성 구조

<표 1>에서는 압축된 데이터에서 제어 비트의 유형을 나타내는데, '00'인 경우는 출력 32비트를 전부 0으로 채운다. 제어비트가 '01'인 경우는 출력비트를 전부 1로 채우게 된다. 그리고 제어비트가 '10'인 경우는 사전에 등록된 데이터로 제어 비트 이후에 사전의 인덱스 값 4비트를 읽어서 사전에 있는 데이터를 출력 값으로 사용한다. 마지막으로 제어비트가 '11'인 경우에는 제어 비트 이후의 32비트를 출력 값으로 사용한다.

〈표 1〉 제어 비트의 종류

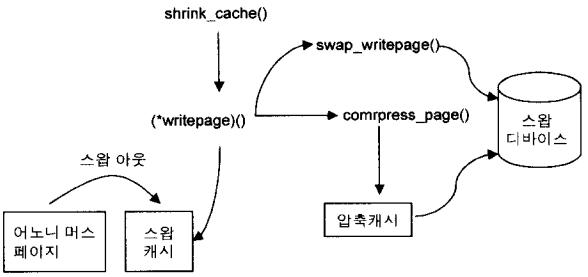
제어 비트	00	01	10	11
제어 비트의 내용	전체 출력으로 0을 채움	전체 출력으로 1을 채움	사전에 등록된 데이터	사전에 등록되지 않은 데이터
제어 비트 이후에 읽을 비트 수	없음	없음	4	32

CAMD 압축 알고리즘의 설계에서 압축률과 수행 속도는 미묘한 관계로 서로 대칭되는 결과를 보여주고 있는데 압축률을 높게 설계할 경우는 코드의 복잡성으로 수행 속도가 떨어지는 문제점이 있고, 압축을 수행하는데 속도를 높이기 위해서는 압축률이 떨어지게 된다. 따라서 이 두 가지를 적절히 조합할 수 있는 단순한 해쉬 함수를 사용하여 적정한 수준을 만족시킬 수 있다.

4. CAMD 압축 알고리즘 구현

CAMD 압축 알고리즘의 동작은 크게 페이지의 움직임에 따라 스왑아웃(Swap Out)과 스왑인(Swap In)으로 이루어진다. 본 논문에서 제안하는 CAMD 압축 알고리즘은 기존 리눅스 커널의 메모리 관리 기능을 수정 및 추가하여 설계한다.

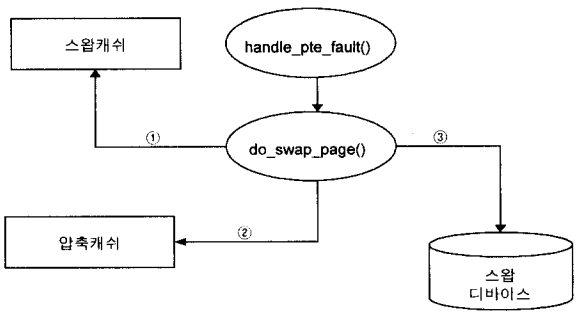
리눅스 운영체제에서 프로세스에 의해 맵핑되지 않는 페이지들을 어노니머스 페이지(anonymous page)라고 한다. 이 어노니머스 페이지들은 더 이상 프로세스에 의해 사용되지 않는다. 그래서 이들 페이지는 시간이 지나게 되면 스왑 캐시 영역으로 스왑 되어진다[7, 8]. 이러한 과정을 그림으로 나타내면 (그림 6)과 같다.



(그림 6) 리눅스 운영체제에서 CAMD 알고리즘과 관련한 스왑 아웃 과정

(그림 6)은 리눅스 운영체제에서 스왑 과정을 보여준다. (그림 6)에서 스왑 아웃(Swap Out)되는 과정은 다음과 같다. 스왑 캐시로 들어오는 페이지들은 shrink_cache()의 (*writepage)() 호출에 의해 스왑 디바이스로 보내진다. (*writepage)() 함수는 swap_writepage() 함수를 호출한다. 스왑 디바이스로의 이동은 swap_writepage()에서 호출하는 rw_swap_page()에 의해 처리된다. 여기서 (*writepage)()의 주소와 연결된 함수 포인터 CAMD_compress_page() 함수의 주소로 연결하면 압축 메모리 시스템에서 스왑 아웃이 이루어진다. 그리고 CAMD_compress_page()에서 압축 캐시 영역의 공간이 부족하게 되면, 기존 커널의 페이지 교체 알고리즘을 이용하여 폐기할 페이지들을 rw_swap_page()를 사용하여 스왑 디바이스로 압축을 풀어서 보낸다.

(그림 6)에서 스왑인 과정은 다음과 같다. 페이지 폴트가 발생하면 (그림 4)에서 보는 바와 같이 스왑 캐시에서 페이지를 찾게 되고 만약 스왑 캐시에서 찾고자 하는 페이지가 없다면 압축 캐시에서 찾는다. 압축 캐시에도 없으면 스왑 디바이스에서 찾게 된다.



(그림 7) 리눅스 운영체제에서 CAMD 알고리즘과 관련한 스왑인 과정

(그림 7)은 CAMD 압축 알고리즘과 관련한 스왑인 과정을 보여준다. 위의 (그림 7)에서 ①번 과정은 스왑 인되는 페이지 중에서 일반 스왑 캐쉬로 쫓겨나는 경우는 나타낸다. ②번 과정은 CAMD 알고리즘이 동작되면서 사용하게 되는 압축 캐쉬로 페이지가 쫓겨나는 경우를 나타낸다. ③번 과정은 일반적인 스왑 과정으로 스왑 디바이스로 페이지가 쫓겨나는 경우를 나타낸다.

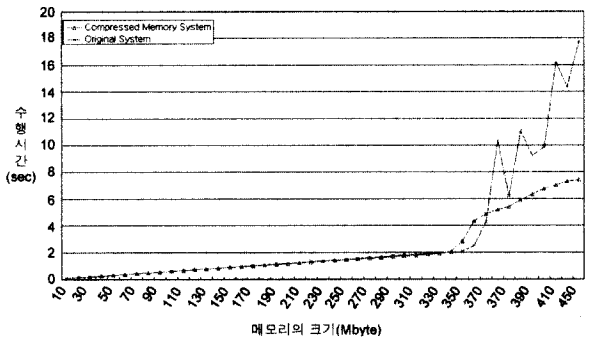
스왑인 요구가 발생하게 되면 리눅스 운영체제의 handle_pte_fault() 함수가 처리를 하게 되는데, 페이지 폴트가 발생하면 호출이 된다. handle_pte_fault() 함수는 do_swap_page() 함수를 호출한다. do_swap_page() 함수는 스왑 캐시 내의 페이지 테이블 엔트리를 검색하게 되고, 만약 찾는 페이지가 발견이 되면 do_swap_page() 함수는 찾은 페이지를 리턴하고 종료를 한다. 스왑 캐시에 찾는 페이지가 없다면 리눅스 커널은 스왑 디바이스를 검색하게 된다. 스왑 디바이스를 검색하기 전에 압축 캐시를 검색하게 하여 CAMD 압축 알고리즘을 적용한다. 찾고자 하는 페이지를 검색할 때 압축된 페이지를 풀어서 검색이 이루어진다.

5. 실험 및 성능 평가

CAMD 가상 메모리 압축 알고리즘에 대한 실험은 리눅스 커널에 이루어졌다. 리눅스 운영체제에 구현된 CAMD 압축 알고리즘을 테스트하기 위해서 물리 메모리를 강제 할당 및 해제를 수행하는 "fillmem"이라는 프로그램을 작성하여 수행하였다. "fillmem" 프로그램을 메모리의 크기별로 수행하여 메모리의 할당과 해제가 일어나면서 스왑핑이 일어날 때의 상태를 검사한다. 이 실험에서 CAMD 압축 알고리즘에 대한 성능 평가는 스왑이 발생할 경우 시스템의 수행 시간과 CPU 사용률을 측정하였다.

CAMD 압축 알고리즘의 동작 수행 시간의 측정은 메모리의 강제 할당과 해제를 수행하는 "fillmem" 프로그램으로 메모리의 크기를 단계적으로 늘리면서 측정하였다. "fillmem" 프로그램에서 메모리의 크기를 처음 10M바이트부터 450M 바이트까지 늘리면서 실험을 진행하였다.

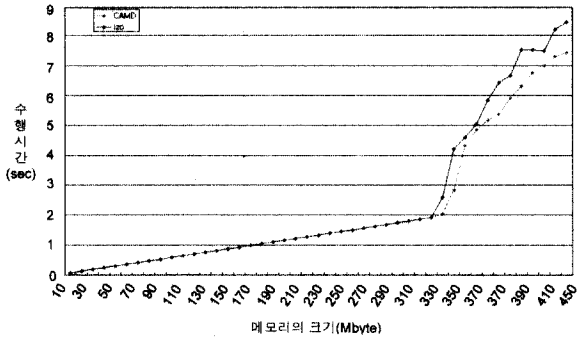
CAMD 알고리즘을 적용하기 위한 압축 메모리 영역의 크기는 8M바이트에서 128M바이트까지 설정하여 두고 실험을 수행한다. 각 압축 메모리 영역의 크기에 대한 실험에서 대표적인 몇가지 경우에 대해서 설명한다.



(그림 8) 압축 메모리 영역의 크기가 64M바이트인 경우 수행 시간 측정

(그림 8)은 CAMD 압축 메모리 영역을 64M바이트로 늘려서 실험한 결과이다. 실험 결과 기존 시스템 보다 CAMD 압축 알고리즘의 수행 시간이 우수하게 나타나는 것을 볼 수가 있다. CAMD 알고리즘의 수행 시간이 최적인 압축 메모리의 크기는 각 시스템의 물리 메모리의 양과 시스템에서 동작중인 프로세스에 의해 사용되는 메모리의 양에 따라 결정이 된다. 이 중에서 메모리 공간 사용에 영향을 많이 미치는 것이 리눅스 운영체제에서 수행하는 데몬 프로세스들이다. 따라서 데몬 프로세스들이 사용하는 메모리의 양에 따라 메모리 공간 사용률 및 성능이 달라질 수가 있다. 본 논문의 실험에서는 전체 물리 메모리의 1/6 정도의 메모리를 CAMD 압축 영역으로 할당할 경우에 최적의 수행 시간을 보이고 있다.

(그림 9)은 기존 LZ 계열의 압축 알고리즘인 LZO 압축 알고리즘과 본 논문에서 제시하는 CAMD 압축 알고리즘과의 수행 속도를 비교한 그래프이다. CAMD 알고리즘이 LZO 알고리즘에 비하여 성능이 우수함을 알 수 있다.



(그림 9) CAMD알고리즘과 LZO알고리즘과의 성능 비교

이상의 실험에서 CAMD 압축 메모리 영역의 크기에 따라 CAMD 압축 알고리즘의 수행 시간이 다르게 나타나는 것을 확인할 수 있다. 압축된 페이지를 저장할 수 있는 공간이 충분하지 못하면 시스템의 수행 시간이 압축 메모리를 사용하지 않은 시스템 보다 좋지 못하게 나타난다.

그리고 CAMD 압축 메모리 영역이 64M 바이트인 경우 가장 수행 시간이 좋게 나타났다. 64M 바이트보다 큰 경우는 압축 메모리 공간을 할당하기 위해 주기억장치의 메모리의 공간을 많이 소모하여 주기억장치에 할당할 수 있는 공간 부족으로 64M 바이트인 경우 보다 수행 시간이 좋지 못하게 나타났다.

CAMD 압축 알고리즘 수행 시간 측정 실험으로 주기억장치의 여유 공간보다 더 많은 메모리의 할당을 요구할 경우에 효율성을 얻을 수 있다는 것을 알 수 있다. 이것은 공통적으로 주기억장치 공간 크기보다 큰 메모리를 할당하고 해제하는 시점에서, 시스템에 스왑이 발생해서 시스템에 부하가 생기기 때문이다. CAMD 가상 메모리 압축 시스템을 적용할 경우 시스템의 주기억장치와 스왑 디바이스와의 접근에서 발생하는 지연 시간을 감소시켜 수행 시간의 효율성을 취할 수 있다.

6. 결 론

본 논문에서는 가상 메모리를 압축할 수 있는 CAMD 압축 알고리즘을 제안한다. CAMD 가상 메모리 압축 알고리즘은 주기억장치 공간을 압축하여 효율적인 공간의 사용을 통해서 시스템 성능을 극대화시킨다. CAMD 압축 알고리즘은 시스템에서 스왑 아웃되는 페이지들을 압축하여 주기억장치 내의 압축 캐시 영역에 저장하는 CAMD 가상 메모리 압축 알고리즘을 설계한다. CAMD 가상 메모리 압축 알고리즘은 페이지의 스왑 아웃되는 시간을 줄이고 스왑 영역의 사용량을 줄임으로써 전체적인 시스템의 메모리 공간을 절약하여 시스템 성능 향상을 기할 수 있다.

본 논문에서 설계한 CAMD 압축 알고리즘은 사전을 기

반으로 한다. CAMD 압축 알고리즘에서는 워드 단위 16개의 버퍼를 잡아서 사전으로 사용한다. 본 논문에서는 페이지 폴트가 일어났을 때 이들 페이지들을 스왑 디바이스로 이동시키지 않고 주기억장치 내의 압축 캐시 영역을 할당하여 CAMD 압축 알고리즘을 사용하여 압축된 페이지를 저장한다면 스왑 디바이스로 이동하는 시간과 횟수를 감소하여 페이지 폴트 응답시간을 줄일 수 있다. 또한, 페이지들의 공간 활용도를 높일 수 있다. 특히 적은 저장 공간을 가진 시스템에 사용하면 효과적인 주 기억장치 관리가 가능하다.

본 논문에서 제시하는 CAMD 가상 메모리 압축 알고리즘은 주기억장치를 압축되지 않은 영역과 압축 캐시 영역으로 나누어서 페이지 폴트가 발생했을 때 느린 스왑 디바이스로 이동하는 페이지들을 압축된 메모리 영역에 저장한다. 그리고 이후에 이 페이지들이 프로세스에 의해 필요로 될 때 느린 스왑 디바이스가 아닌 압축 메모리 영역에서 데이터를 가지고 옴으로써 보조기억장치로의 접근 횟수를 줄일 수 있도록 설계하였으며 압축 스왑 캐쉬를 사용함으로써 보조기억장치의 저장 공간을 늘리는 효과를 낼 수 있다.

본 논문의 실험에서 가상 메모리 압축 알고리즘을 적용한 시스템과 적용하지 않은 시스템과의 성능 비교에서 주기억장치의 가용 공간보다 더 많은 메모리를 요구하는 작업에서 수행 시간의 효율성을 얻을 수 있다. 이것은 시스템에서 스와핑이 발생할 때의 지연시간을 최소화 하여 전체적인 수행 시간의 향상을 얻을 수 있다. 본 논문에서 제시한 압축 알고리즘은 전체 비트가 0과 1로 구성된 데이터를 적은 비트로 압축하고 사전에 등록된 데이터와 사전에 등록되지 않은 데이터들을 제어 비트에 의해 처리되도록 구현한다. 그리고 사전의 운영은 해쉬를 사용하여 구현하였다.

참 고 문 헌

[1] Caroline Benveniste, "Cache-Memory Interfaces in Compressed Memory System," IEEE, 2001.

[2] Dong Banks, Mark Stemm, "Investigating Virtual Memory Compression on Portable Architectures," 1995.

[3] Fred Douglis, "The compression Cache : Using on-line compression to extend physical memory," USENIX Proceedings, Winter, 1993.

[4] S. Kwong and Y. F. Ho, "A statitical Lempel-Ziv Compression Algorithm for Personal Digital Assistant(PDA)," IEEE, 2001.

[5] Ian McDonald, "The use of a Compressed Cache in an Operating System supporting Self-Paging," September, 1999.

[6] Sumit Roy, Raj Kumar, Milos Prvulovic, "Improving System Performance with Compressed Memory," IEEE, 2001.

[7] David A Rusling, "The Linux Kernel," January, 1999.

[8] M. S. Pinho, W. A Finamore, "Using arithmetic code to improve performance of Lempel-Ziv encoders," Electronic Letters, Aug., 2000.

[9] Ross N. Williams, "An Extremely Fast Ziv-Lempel Data Compression Algorithm," Data Compression conference, 1991.

[10] Maurice J. Bach, "The Design of The Unix Operating System," Prentice-Hall International Editions.

[11] Khalid Sayood, "Introduction to Data Compression Second Edition," Morgan Kaufmann, 2000.

[12] Data Compression Reference Center http://www.rasip.fer.hr/research/compress/algorithms/index.htm.

[13] Ross N. Williams, "An Extremely Fast Ziv-Lempel Data Compression Algorithm," Data Compression conference, 1991.

[14] Fred Douglis, "The compression cache : Using on-line compression to extend physical memory," USENIX Proceedings, Winter, 1993.

[15] Ian McDonald, "The use of a Compressed Cache in an Operating System supporting Self-Paging," September, 1999.

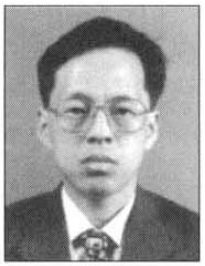
[16] Ross N. Williams, "An Extremely Fast Ziv-Lempel Data Compression Algorithm," Data Compression conference, 1991.

[17] S. Kwong and Y. F. Ho, "A statitical Lempel-Ziv Compression Algorithm for Personal Digital Assistant(PDA)," IEEE, 2001.

[18] Scott F. Kaplan, "Compressed Caching and Modern Virtual Memory," 1999.

[19] Caroline Benveniste, "Cache-Memory Interfaces in Compressed Memory System," IEEE, 2001.

[20] Paul R. Wilson, Scott F. Kaplan, and Yannis Smaragdakis, "The Case for Compressed Caching in Virtual Memory Systems" Proceedings of the Usenix, 1999.



장 승 주

e-mail : sjjang@deu.ac.kr

1985년 부산대학교 계산통계학과(전산학) 학사

1991년 부산대학교 계산통계학과(전산학) 석사

1996년 부산대학교 컴퓨터공학과 박사

1987년~1996년 한국전자통신연구원 시스템 S/W연구실

1993년~1996년 부산대학교 시간강사

2000년~2002년 University of Missouri at Kansas City, visiting professor

1996년~현재 동의대학교 컴퓨터공학과 부교수

관심분야 : 운영체제, 분산시스템, Active Network, 시스템 보안, 보안 정형 기법