한국컴퓨터정보학회 동계학술대회 논문집 제27권 제1호 (2019. 1)

리눅스 기반 시스템의 페이지 초기화 지연 단축을 위한 향상된 캐시-핫 페이지 할당 기법

양석우[O], 노순현[*], 홍성수[*]
[O*]서울대학교 전기•정보공학과
e-mail: {swyang[O], shnoh[*], sshong[*]}@redwood.snu.ac.kr

Improved Cache-hot Page Allocation Technique for Reducing Page Initialization Latency of Linux Based Systems

Seokwoo Yang[O], Sunhyeon Noh[*], Seongsoo Hong[*]
[O*]Dept. of Electrical and Computer Engineering, Seoul National University

● 요 약 ●

최근 사용자 대화형(user-interactive) 응용들은 OS에게 많은 양의 메모리를 빈번하게 요구한다는 특징을 보인다. 응용의 메모리 할당 요청이 발생하면 OS는 할당할 페이지의 초기화 작업을 필수적으로 수행하는데, 빈번하게 발생하는 페이지 초기화 작업이 응용의 성능을 저하시키고 있다. 기존 리눅스 기반 시스템은 페이지 초기화 지연을 단축하기 위해 CPU의 캐시에 매핑되어 있어서 초기 값을 빠르게 쓸 수 있는 페이지인 캐시-핫(cache-hot) 페이지를 우선적으로 할당한다. 하지만 기존 리눅스는 각 코어별로 캐시-핫 페이지를 인식하고 관리하며, 다른 코어가 관리하는 캐시-핫 페이지에는 접근할 수 없다. 이러한 정책 때문에 다른 코어가 공유 캐시(shared cache)에 매핑된 캐시-핫 페이지를 관리하고 있더라도, 이를 할당받지 못하고 캐시-콜드(cache-cold) 페이지를 할당받는 경우가 발생한다. 본 논문에서는 공유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 별도로 인식하고 공유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 모든 코어가 활용할 수 있게 하여, 응용이 캐시-핫 페이지를 할당받을 확률을 기존 기법보다 높이는 향상된 캐시-핫 페이지 할당 기법을 제안한다. 제안된 기법은 페이지 할당 요청이 발생하면 먼저 각 코어의 사유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 우선적으로 할당하고, 할당에 실패하면 공유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 할당한다. 이를 통해 캐시-핫 페이지를 할당받을 확률을 기존 기법보다 높이고, 결과적으로 평균 페이지 초기화 지연을 단축한다. 제안된 기법을 리눅스 커널 4.18.10버전 기반 환경에서 구현하여 실험한 결과, 평균 페이지 초기화 지연이 기존 리눅스 시스템과 비교하여 약 7% 단축되었다.

키워드: 리눅스(Linux), 캐시(Cache), 페이지 초기화(Page Initialization)

I. Introduction

현대 컴퓨팅 플랫폼의 메모리 용량 증가에 따라, 메모리 집약적(memory intensive)인 응용들이 많이 등장하게 되었다. 그 중에서도 특히 사용자 대화형 응용은 사용자의 입력이 발생하면, 이를 처리하기 위해 OS에게 빈번하게 큰 메모리 공간의 할당을 요청한다는 특징이 있다.

이러한 특징 때문에 OS에서 메모리 공간 할당 요청을 처리하기 위해 발생시키는 마이너 페이지 폴트의 처리시간이 사용자 대화형 응용의 수행시간에서 큰 비중을 차지하게 되었다. 마이너 페이지 폴트 핸들러(handler)는 커널의 메모리 관리자로부터 비어있는 페이지를 할당받아 응용의 가상 메모리 공간에 매핑하여, 응용의 메모리 공간 할당 요청에 대응하는데, Table. 1.은 이러한 마이너페이지 폴트의 처리시간이 응용의 수행시간에서 차지하는 비율을 대표적인 사용자 대화형 응용인 웹 브라우저에서 측정한 것이다. 측정 결과 초당 약 5만~10만회의 마이너 페이지 폴트가 발생하였고, 전체 수행시간 중 마이너 페이지 폴트 처리 지연의 비율이 약 11~22%로 큰 비중을 차지함을 확인하였다.

Table 1. 응용의 수행시간에 마이너 폴트가 미치는 영향

측정에 사용한 응용	초당 마이너 페이지 폴트 발생 횟수(평균)	전체 수행시간 중 마이너 페이지 폴트 처리 시간의 비율
WebGL-based simulation[1]	48599회	22%
GoogleMap[2]	80654회	11%
Naver지도[3]	97941회	12%

이 때 마이너 페이지 폴트 처리 과정에서 OS는 페이지 초기화를 수행한다. 페이지 초기화는 마이너 페이지 폴트 핸들러가 사용자 태스크에게 제공할 페이지에 0을 채워넣는 작업으로, 과거에 해당 페이지를 사용한 태스크나 커널의 데이터가 유출되는 것을 방지하기 위한 과정이다. 기존 연구에 따르면 이러한 마이너 페이지 폴트 처리 지연 중 페이지 초기화 작업에 소요되는 시간이 큰 비중을 차지한다. 하나의 마이너 페이지 폴트 처리가 약 3400 사이클이 소요될 때, 페이지 초기화에 최대 약 1400 사이클이 소요되어, 마이너 페이지 폴트 처리 시간의 약 40%를 차지하였다[4].

이러한 페이지 초기화를 빠르게 수행하기 위한 대표적인 방법으로 캐시-핫 페이지 활용이 있다. 캐시-핫 페이지는 페이지의 일부 또는 전체가 CPU의 캐시에 매핑되어 있는 페이지다. DRAM보다 캐시에 접근하는 속도가 훨씬 빠르기 때문에 페이지가 캐시에 매핑되어 있다면 페이지가 DRAM에 있을 때 보다 페이지에 초기 값을 쓰는데 소요되는 시간이 짧다. 따라서 캐시-핫 페이지는 캐시에 매핑되어있지 않은 캐시-콜드 페이지에 비해 초기화를 빠르게 수행할 수 있다.

리눅스는 이러한 사실에 기반 하여 마이너 폴트 발생 시 캐시-핫 페이지를 할당하여 페이지 초기화 지연을 단축한다. 리눅스의 캐시-핫 페이지 할당 기법은 각 코어에서 수행중인 태스크가 할당을 해제하는 페이지의 캐시-핫 여부를 판단하고, 캐시-핫 페이지로 추정되는 페이지는 각 코어의 캐시-핫 페이지 리스트에서 관리된다. 각 코어는 자신의 캐시-핫 페이지 리스트를 통해서만 캐시-핫 페이지를 할당받을 수 있다.

하지만 이러한 정책은 뚜렷한 한계점을 가진다. CPU의 공유 캐시에 매핑된 캐시-핫 페이지는 모든 코어에서 짧은 접근시간으로 접근할 수 있다. 하지만 기존 리눅스의 캐시-핫 페이지 할당 정책은 페이지 할당 요청이 발생한 캐시-핫 페이지 리스트가 비어있는 경우에 다른 코어의 캐시-핫 페이지 리스트에 공유 캐시에 매핑된 캐시-핫 페이지가 존재해도, 이를 할당받지 못한다. 따라서 리눅스는 공유 캐시에 매핑된 캐시-핫 페이지를 잘 활용하지 못하고 있다. 대부분의 현대 CPU들은 사유 캐시의 수십에서 수백 배 크기의 공유 캐시를 가지고 있으므로, 공유 캐시에 매핑된 캐시-핫 페이지들을 잘 활용하지 못하는 것은 치명적인 한계점이다.

본 논문에서는 기존 기법의 이러한 문제를 해결하기 사유 캐시 또는 공유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 각각 인식하고, 공유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 모든 코어가 활용할 수 있게 하여 응용이 캐시-핫 페이지를 할당받을 확률을 기존 기법보다 높이는 향상된 캐시-핫 페이지 할당 기법을 제안한다. 제안된 기법은 페이지 할당 요청이 발생하면 먼저 각 코어의 사유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 우선적으로 할당한다. 만약 사유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지가 없다면 캐시-콜드 페이지를 할당받는 대신 공유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 할당한다. 이를 통해 캐시-핫 페이지를 할당받을 확률을 기존 기법보다 높이고, 결과적으로 평균 페이지 초기화 지연을 단축한다.

우리는 제안된 기법을 리눅스 커널 4.18.10버전 기반 데스크탑 환경에서 구현하고, 평균 페이지 초기화 지연을 측정하였다. 실험 결과 기존의 커널에 비하여 제안된 기법이 적용된 커널에서 평균 페이지 초기화 지연이 약 7% 단축되었다.

II. Problem Statement

본 논문의 목적은 평균 페이지 초기화 지연을 단축하는 것이다. 페이지 초기화 지연은 마이너 페이지 폴트 처리 시 페이지의 시작 주소부터 마지막 주소까지 초기 값을 입력하는데 소요되는 지연이다. 평균 페이지 초기화 지연은 응용을 수행하는 동안 발생한 모든 페이지 초기화 지연의 평균값이다.

페이지 초기화 지연은 다음과 같은 요소로 구성된다. 1) DRAM에 접근하여 목표 메모리 영역을 캐시로 적재하는 지연, 2) 캐시에 적재된 영역에 초기 값을 입력하는 지연이다. CPU에서 DRAM에 접근하는 시간은 캐시에 접근하는 시간에 비해 매우 길기 때문에 1)에 해당하는 지연이 페이지 초기화 지연의 대부분을 차지한다. 캐시-핫 페이지 할당을 통해 1)에 해당하는 지연을 단축할 수 있다.

본 논문에서는 평균 페이지 초기화 지연 중에서 1)에 해당하는 지연을 줄이기 위해 캐시-핫 페이지를 할당받을 확률을 기존 리눅스 커널보다 높인다. 페이지 할당 요청 발생 시 캐시-핫 페이지를 할당받는 비율이 높을수록, 평균 페이지 초기화 지연에서 1)에 해당하는 지연이 단축된다.

III. The Proposed Solution

본 장에서는 제안된 기법을 설명한다. 제안된 기법은 캐시-핫 페이지를 할당받을 확률을 높여 평균 페이지 초기화 지연을 단축하기 위해 사유 캐시와 공유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 각각 인식하고 공유 캐시에 매핑된 것으로 추정되는 캐시-핫 페이지를 모든 코어가 활용할 수 있도록 한다.

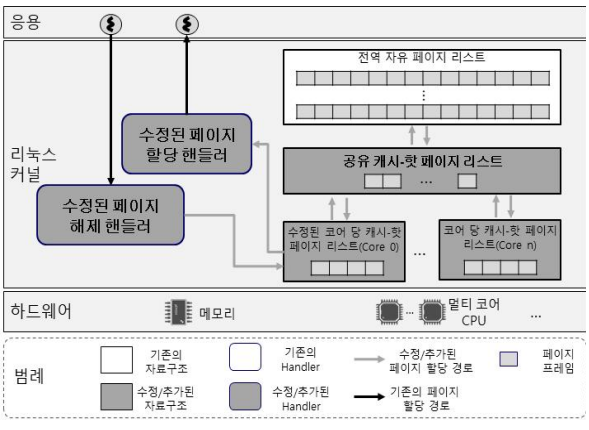

Fig. 1. 제안된 기법 개요

Fig. 1.은 제안된 기법의 구조를 나타낸다. 제안된 기법을 위해 리눅스의 물리 메모리 할당 메커니즘에 새로운 구성요소를 추가하거나, 일부 구성요소를 수정하였다. 각각은 다음과 같다. 1) 수정된 코어 당 캐시-핫 페이지 리스트, 2) 공유 캐시-핫 페이지 리스트, 3) 수정된 페이지 할당 핸들러, 4) 수정된 페이지 해제 핸들러이다.

수정된 코어 당 캐시-핫 페이지 리스트는 사유 캐시에 매핑되어 있을 것으로 추정되는 페이지들의 연결 리스트이다. 제안된 기법은 항상 이 리스트의 선두에 사유 캐시에서 가장 캐시-핫한 페이지를 위치시킨다. 기존의 코어 당 캐시-핫 페이지 리스트의 *크기*는 시스템의 메모리 용량을 고려하여 정해졌지만, 수정된 코어 당 캐시-핫 페이지 리스트는 CPU의 사유 캐시의 *크기*를 고려하여 정해진다. 제안된 기법은 이를 통해 수정된 코어 당 캐시-핫 페이지 리스트의 최후위에 존재하는 페이지는 사유 캐시보다는 공유 캐시에 매핑되어있을 확률이 높다고 간주한다.

공유 캐시-핫 페이지 리스트는 공유 캐시에 매핑되어있을 것으로 추정되는 페이지들의 연결 리스트로 구성된다. 제안된 기법은 항상 이 리스트의 선두에 공유 캐시에서 가장 캐시-핫한 페이지를 위치시킨다. 공유 캐시-핫 페이지 리스트는 현재 리스트의 *크기*, 최대 리스트의 *크기* 값을 가지며, 최대 리스트의 *크기*는 CPU의 공유 캐시의 *크기*를 고려하여 정해진다. 이를 통해 제안된 기법은 공유 캐시-핫 페이지 리스트의 최후위에 위치한 페이지는 캐시-콜드 페이지로 간주한다.

수정된 페이지 할당 및 해제 핸들러는 수정된 코어 당 캐시-핫 페이지 리스트와 공유 캐시-핫 페이지 리스트에 기반하여 페이지를 할당하고 해제하기 위해 기존의 페이지 할당 및 해제 핸들러를 수정한 것이다. 수정된 핸들러들은 수정된 코어 당 캐시-핫 페이지 리스트, 공유 캐시-핫 페이지 리스트, 전역 자유 페이지 리스트 순서로 페이지의 할당과 해제를 시도한다.

본 장의 나머지 부분에서는 이러한 구성요소를 사용한 새로운 페이지 할당 및 해제 동작을 설명한다.

1. Page Deallocation

페이지 해제 동작 시 제안된 기법의 목표는 코어 당 캐시-핫 페이지 리스트와 공유 캐시-핫 페이지 리스트에서 각각 사유 캐시와 공유 캐시에서 캐시-핫할 확률이 높은 페이지일수록 리스트의 선두에 위치하도록 정렬하는 하는 것이다.

이를 위해 페이지 해제 핸들러는 페이지 해제 요청이 발생하면 우선 코어 당 캐시-핫 페이지 리스트의 선두에 해제된 페이지를 삽입한다. 가장 최근에 해제된 페이지일수록 코어의 사유-캐시에서 캐시-핫할 확률이 높기 때문이다.

이 때 코어 당 캐시-핫 페이지 리스트가 가득 찼다면, 페이지 해제 핸들러는 코어 당 캐시-핫 페이지 리스트의 가장 후위에 존재하는 페이지들부터 공유 캐시-핫 페이지 리스트의 선두로 삽입한다. 이 페이지들은 코어의 사유 캐시에 매핑되어 있을 확률은 낮지만, 공유 캐시에 매핑되어 있을 가능성이 높다. 따라서 이 페이지들을 전역 자유 페이지 리스트로 보내는 대신 공유 캐시-핫 페이지 리스트에 보관하여 페이지 할당 요청 시 활용할 수 있다.

만약 공유 캐시-핫 페이지 리스트도 가득 찼다면, 페이지 해제 핸들러는 공유 캐시-핫 페이지 리스트의 가장 후위에 있는 페이지들부터 전역 자유 페이지 리스트로 반환한다. 이 페이지들은 더 이상 CPU의 어떤 캐시에도 매핑되어 있지 않을 확률이 높기 때문에 캐시-콜드 페이지로 간주한다.

2. Page Allocation

페이지 할당 동작 시 제안된 기법의 목표는 시스템에 존재하는 캐시-핫 페이지들 중 해당 코어에서 가장 페이지 초기화 지연을 단축할 수 있는 페이지를 할당받는 것이다.

이를 위해 페이지 할당 핸들러는 페이지 할당 요청이 발생하면 우선 코어 당 캐시-핫 페이지 리스트에서 페이지 할당을 시도한다. 코어 당 캐시-핫 페이지 리스트에 페이지가 존재하면, 페이지 할당 핸들러는 리스트에서 가장 캐시-핫한 페이지를 할당받기 위해 리스트의 선두에 위치한 페이지를 할당받는다.

만약 코어 당 캐시-핫 페이지 리스트에 페이지가 존재하지 않으면 페이지 할당 핸들러는 공유 캐시-핫 페이지 리스트에서 캐시-핫 페이지 할당을 시도한다. 마찬가지로 공유 캐시에서 가장 캐시-핫한 페이지를 할당받기 위해, 리스트의 선두에 위치한 페이지를 할당받는다. 이를 통해 각 코어의 사유 캐시뿐만 아니라 공유 캐시에 매핑된 캐시-핫 페이지들을 할당받을 수 있으므로 캐시-핫 페이지를 할당받을 확률이 높아진다. 만약 공유 캐시-핫 페이지 리스트도 비어있다면, 페이지 할당 핸들러는 전역 자유 페이지 리스트에서 캐시-콜드 페이지를 할당받는다.

IV. Experimental Evaluation

1. 실험 설계

Table 2. 실험 환경

HW	CPU	Intel i7-6700
	Memory	DDR3 8GB DRAM
SW	Kernel	Linux Kernel version 4.18.10
	OS	Ubuntu 16.04

본 장에서는 제안된 기법을 Table. 2.와 같은 환경에서 검증한 결과를 설명한다. 제안된 기법의 검증을 위해 두 가지 실험을 진행하였다. 각각은 1) 평균 페이지 초기화 지연의 단축을 확인하기 위한 실험, 2) 제안된 기법의 런타임 오버헤드를 측정하기 위한 실험이다. 두 실험은 모두 워크로드로 Basemark Web3.0[11]을 사용하였다. Basemark Web 3.0은 대표적인 사용자 대화형 응용인 웹 브라우저의 성능을 평가하기 위해 사용된다.

2. 평균 페이지 초기화 지연 측정 실험

이 실험의 목적은 제안된 기법의 적용을 통해 평균 페이지 초기화 지연의 개선을 파악하는 것이다. 평균 페이지 초기화 지연의 개선 정도를 측정하기 위해, 우리는 기존 리눅스 커널과 제안된 기법이 적용된 리눅스 커널에서 각각 워크로드를 10회 수행하여 평균 페이지 초기화 지연을 측정하였다.

Fig. 2.는 실험 결과를 나타낸다. 제안된 기법이 적용된 리눅스 커널에서 기존 커널에 비해 평균 페이지 초기화 지연이 약 7% 개선되었다.

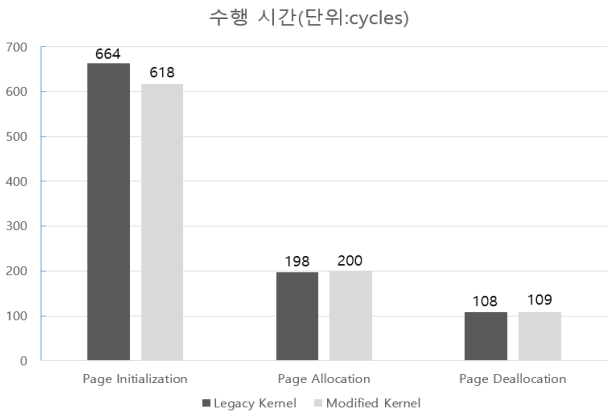

Fig. 2. 평균 페이지 초기화 지연 및 오버헤드 측정 결과

3. 오버헤드 측정 실험

이 실험의 목적은 제안된 기법에 의한 런타임 오버헤드가 미미함을 보이는 것이다. 런타임 오버헤드의 측정을 위해 우리는 기존 리눅스 커널과 제안된 기법이 적용된 리눅스 커널에서 각각 워크로드를 10회 수행하여 페이지 할당 및 해제 핸들러의 동작 시간을 측정하였다. 이를 통해 페이지 할당 및 해제 과정에서 기존 리눅스 커널에 비해 제안된 기법이 가지는 오버헤드를 파악한다. Fig. 2.는 실험 결과로, 기존 리눅스 커널에 비해 제안된 기법이 적용된 커널의 페이지 할당 및 해제에 소요되는 시간이 약 1% 증가하여, 오버헤드가 매우 미미함을 알 수 있다.

V. Conclusions

본 논문은 페이지 초기화 지연을 단축하기 위해 다중 레벨 캐시-핫 페이지 리스트를 통한 캐시-핫 페이지 할당 기법을 제안하였다. 제안된 기법은 CPU의 공유 캐시에서 캐시-핫한 페이지를 인식하기 위한 자료구조를 추가하여, 기존 리눅스 커널에 비해 캐시-핫 페이지를 할당받을 확률을 높였다. 이를 통해 평균 페이지 초기화 지연을 기존 리눅스 커널과 비교하여 약 7% 단축하였다.

REFERENCES

[1] WebGL Samples, “Jellyfish simulation”, https://arodic.github.io/p/jellyfish/

[2] Google Maps, https://maps.google.com/

[3] Naver Maps, https://map.naver.com/

[4] Valat, Sebastien, Marc Perache, and William Jalby. "Introducing kernel-level page reuse for high performance computing." Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness. ACM, 2013.

[5] Seshadri, Vivek, et al. "RowClone: fast and energy efficient in-DRAM bulk data copy and initialization." Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture. ACM, 2013.

[6] Jiang, Xiaowei, et al. "Architecture support for improving bulk memory copying and initialization performance." 2009 18th International Conference on Parallel Architectures and Compilation Techniques. IEEE, 2009.

[7] FreeBSD Documents, "Pre-faulting and zeroing optimizations",https://www.freebsd.org/doc/en_US.ISO8859-1/articles/vm-design/prefault-optimizations.html

[9] DragonflyBSD, “No more page zeroing”, https://www.dragonflydigest.com/2016/08/04/18484.html

[10] Linux kernel documents, “Physical page allocation”, https://www.kernel.org/doc/gorman/html/understand/understand009.html

[11] Basemark Web3.0, https://web.basemark.com/