저작자표시-비영리-변경금지 2.0 대한민국

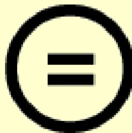
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

권석민 석사 학위논문

# 휴즈 페이지에 대한 성능 분석 및 최적화 연구

# Workload Analysis and Optimization for Efficient use of Huge Page

2020년 12월

서울대학교 대학원

컴퓨터공학부

권석민

# 휴즈 페이지에 대한 성능 분석 및 최적화 연구

# Workload Analysis and Optimization for Efficient use of Huge Page

지도교수 염헌영

이 논문을 석사 학위논문으로 제출함

2020년 12월


서울대학교 대학원

컴퓨터 공학부

권석민


권석민의 석사 학위논문을 인준함

2021년 1월

위 원 장 ________ 장병탁 (인)

부위원장 ________ 염헌영 (인)

위　　원 ________ 엄현상 (인)

# 초록

오늘날 테라바이트 이상의 메모리를 지원하는 컴퓨팅 플랫폼과 이러한 대용량 메모리를 활용하는 워크로드들이 일반화되고 있다. 이러한 대용량 메모리의 사용은 주소 변환 오버헤드와 TLB miss가 큰 제약 사항이다. 이를 해결하기 위한 방안으로 Huge Page를 사용할 수 있다. 하지만 Huge Page로 인한 메모리 낭비로 인해 명확한 성능상의 이점에도 불구하고 대부분의 경우 Huge Page를 시스템 상에서 비활성화할 것을 권장하고있다. 이에 본 논문에서는 실행 시간과 최대 메모리 사용량 분석을 통해 다양한 워크로드에서 Huge Page를 사용했을 때 Huge Page를 효율적으로 적용할 수 있는 워크로드를 선택했다. 그런 다음 memory access tracing 도구를 이용하여 해당 워크로드의 메모리 접근 패턴을 분석하여 Huge Page 보다 효율적으로 사용할 수 있는 방법을 탐색했다. 반복적으로 접근되는 500MB 이상의 크기를 가지는 데이터 구조에 해당하는 메모리 영역에 대해서만 부분적으로 Huge Page를 적용하여 실행 시간과 최대 메모리 사용량 사이의 적절한 trade-off를 결정함으로써 해당 워크로드에서 Huge Page를 최적으로 적용하였다.

주요어 : hugepage, thp, memory management

학번 : 2018-26707

# 목차


초록 i

목차 ii

그림 목차 iii

제 1 장 서론 1

제 2 장 배경 2

2.1 Virtual Memory Trend 2

2.2 Huge Pages 2

2.3 Memory Access Tracing Technique 2

제 3 장 워크로드 분석 3

3.1 Huge Page 사용에 따른 실행 시간 3

3.2 Huge Page 사용에 따른 최대 메모리 사용량 3

3.3 워크로드 분석 결과 및 선택 3

제 4 장 설계 및 구현 4

4.1 Memory Access Pattern 확인 4

4.2 효율적인 Huge Page 적용을 위한 조건 탐색 4

4.3 선택적 Huge Page 적용 4

제 5 장 실험 및 분석 5

5.1 ocean_ncp 실행 시간 5

5.2 ocean_ncp 최대 메모리 사용량 5

제 6 장 관련 연구 6

제 7 장 결론 7

참고문헌 8

ABSTRACT 23

# 그림 목차

그림 3.1 parsec3 워크로드 실행시간 7
그림 3.2 splash2x 워크로드 실행시간 8
그림 3.3 parsec3 워크로드 최대 메모리 사용량 10
그림 3.4 splash2x 워크로드 최대 메모리 사용량 11
그림 4.1 ocean_ncp의 메모리 접근 패턴 13
그림 4.2 ocean_ncp에서 반복적으로 접근되는 메모리 영역 17
그림 4.3 ocean_ncp의 자료구조 18
그림 4.4 0번부터 1024번까지의 메모리 접근 패턴
그림 4.5 Huge Page의 선택적 사용
그림 5.1 단계별 Huge Page 선택 사용에 따른 실행시간
그림 5.2 단계별 Huge Page 선택 사용에 따른 최대 메모리 사용량

# 제 1 장 서론

오늘날 최신의 컴퓨팅 플랫폼은 테라바이트 이상의 메모리를 지원하며 이러한 대용량 메모리를 활용하는 워크로드들 또한 일반화되고 있다. [1] 하지만 이러한 대용량 메모리의 사용은 주소 변환이 큰 제약 사항이다. 현대의 모든 프로세서들은 주소 변환을 위해 페이지 테이블을 사용하고, 가상 주소와 물리적 주소의 매핑을 캐싱하기 위해 TLB 를 사용하는데 TLB 의 용량은 DRAM 에 상응하는 속도로 확장시킬 수 없다. 그러므로 대용량 메모리 워크로드가 기존의 normal page 의 크기(즉, 4KB)를 사용하는 경우 주소 변환과 TLB 미스로 인해 성능이 심각하게 저하될 수 있다. AMD 의 다차원 페이지 테이블에서는 최악의 경우 주소 변환 비용이 6 배까지 증가한다. [2]

하드웨어 제조업체들은 Huge Page 에 대한 더 나은 지원을 통해 TLB 미스를 줄여 주소 변환 오버헤드를 감소시켰다. 하지만 이는 운영 체제가 Huge Page 를 관리하는 능력에 따라 달라진다. 최근 더 나은 Huge Pgae 지원에 대한 오버헤드가 하드웨어에서 시스템 소프트웨어로 이행되었다.(예를 들어, 인텔의 Skylake [3]) 그렇지만 운영 체제의 메모리 관리는 일반적으로 4KB 의 normal page 에 초점을 두고 있기때문에 Huge Page 를 사용할 때 다양한 워크로드에서 허용될 수 없는 성능 오버헤드, 성능 가변성, 메모리의 낭비를 야기한다. 이러한 점들은 Huge Page 를 사용할 경우 평균적인 경우에 명백한 성능상의 이점이 있음에도 불구하고 시스템 관리자가 Huge Page 를 일반적으로 비활성화할 정도로 심각하고 흔한 문제이다. (예: SAP, Splunk, Redis, MongoDB, Couchbase 등) [4, 5, 6, 7, 8]

본 논문에서는 다양한 워크로드에서 Huge Page를 사용함으로써 발생할 수 있는 문제점을 최소화하여 효과적인 Huge Page 이용을 도모한다. 정해진 조건을 충족하는 메모리 영역에 대해서만 Huge Page를 사용하게 함으로써 메모리 내부 조각화로 인해 낭비되는 메모리는 줄이고, 성능상의 이점은 획득할 수 있게 한다.

효율적인 Huge Page 사용을 위한 방법을 적용하기 위해 기존 워크로드에서 normal page를 사용했을 때와 Huge Page를 사용했을 때의 성능과 최대 메모리 사용량을 비교 분석하고, 보다 최적화된 Huge Page 사용을 위한 코드를 설계하고 구현한다.

# 제 2 장 배경

## 2.1 Virtual Memory Trend ; 가상 메모리 경향

가상 메모리는 프로그램에서 사용하는 주소 공간을 물리적 메모리로부터 분리시킨다. 최근에 사용한 페이지 테이블 엔트리들이 TLB 에 캐싱되고, 페이지 테이블은 가상 페이지 번호를 물리적 페이지 번호에 매핑시킨다. 페이지 크기를 증가시키면 TLB reach(TLB 에 캐싱된 주소 변환 영역에 포함되는 데이터의 크기)가 증가하지만 페이지의 크기가 커질수록 더 큰 연속적인 물리적 메모리를 필요로 한다. Huge Page 는 메모리 내부 조각화(할당 받은 메모리 영역 내에서 사용되지 않는 부분)가 발생할 수 있으며 Huge Page 이외의 다른 연속적인 물리적 메모리 할당을 제한하여 메모리 외부 조각화도 증가시킬 수 있다. 2.1 에서는 Huge Page 의 시스템 지원을 필요로 하는 최신의 하드웨어 경향을 살펴본다.

첫 번째로는 DRAM 크기의 증가이다. DRAM 의 크기가 커지면 페이지 테이블의 레벨이 더욱 증가하고, 이는 가상 페이지 번호를 찾는데 필요한 메모리 참조 횟수가 증가함을 의미한다. x86 에서는 하나의 주소 변환 수행시 최악의 경우 4 레벨 페이지 테이블을 사용해야 한다. 두 번째로는 TLB reach 의 증가이다. 최근 인텔은 TLB 를 2 레벨로 전환했으며 지난 몇 년 동안 Huge Page 를 위한 2 단계 TLB 엔트리의 개수를 Haswell[9]에 대해서는 1024 개, Skylake[3]에 대해서는 1536 개 까지 지원했다. 이러한 최신 하드웨어 경향을 Huge Page 사용의 강력한 동기가 된다.

## 2.2 Huge Pages

페이지 테이블 워크 오버헤드를 최소화하기 위해 TLB는 이전의 주소 변환 결과를 캐싱한다. 하지만 워킹 셋이 TLB의 크기보다 훨씬 커지게 되면 TLB를 사용하는 장점이 상쇄된다. 이러한 TLB cache flooding의 잘 알려진 해결책 중 대표적인 것이 Huge Page이다. 서로 다른 플랫폼들은 서로 다른 크기의 Huge Page를 지원한다. x86-64 머신에서는 normal page의 크기는 4KB이고, 2MB, 1MB 크기의 Huge Page를 지원한다. (별도의 언급이 없는 한은 2MB 크기의 Huge Page가 1GB 크기의 Huge Page보다 대부분의 경우에서 더 유용하기 때문에 Huge Page의 크기는 2MB로 한다.) 이러한 Huge Page는 normal page보다 2배 이상의 크기를 가지기 때문에 TLB reach가 증가하여 TLB miss 수를 줄일 수 있다. 오늘날 Big Data, Cloud System, Machine Learning 등의 최신 워크로드들은 낮은 지역성을 가지며 많은 메모리를 사용하는 특징을 가지므로 Huge Page의 이용이 더욱 중요해졌다. 대부분의 클라우드 컴퓨팅 시스템 환경에서의 일반적인 설정인 확장된 페이지 테이블을 사용하는 가상 메모리 환경에서는 페이지 테이블 워크 비용이 훨씬 증가하고 있다. 최근 인텔은 메모리 크기가 큰 시스템의 경우 페이지 테이블의 레벨을 늘리려고 하고 있다. 이러한 최신의 경향들은 Huge Page 사용의 중요성을 강하게 시사하고 있다. [10]

리눅스에서는 두 가지 방법으로 Huge Page를 지원한다. 첫 번째는 hugetlbfs 방식이다. 이 방식은 프로그램이 Huge Page를 명시적으로 요청할 때 사용할 수 있는 기존의 방법이다. 시스템은 큰 크기의 연속된 메모리 영역을 예약하고 해당 영역을 Huge Page를 위한 pool로 활용한다. hugetlbfs는 그 나름의 유용성을 입증해왔지만 Huge Page 이용을 위해 프로그래머의 추가적인 코드 수정이 필요하다. 그리고 만약에 Huge Page가 시스템이 Huge

Page 를 위해 예약한 메모리 영역을 완전히 활용하지 못한다면 예약한 메모리 영역은 메모리 낭비를 발생시킨다. 리눅스에서 Huge Page 를 지원하는 두 번째 방식은 THP;Transparent Huge Page 이다. 이 방식은 비교적 최근의 새로운 방식이다. 이름처럼 투명하게 동작하기 때문에 추가적인 변경없이 Huge Page 의 활용이 손쉽게 가능하다. 시스템은 normal page 를 Huge Page 로 변환하거나 반대로 Hug Page 를 normal page 로 자동 변환한다. 이 과정은 페이지 폴트 핸들러와 khugepaged 커널 스레드 데몬에 의해 수행된다. 페이지 폴트 핸들러는 폴트가 난 메모리 영역에 대해 normal page 와 huge page 중 어떤 페이지를 쓸지 결정하게 된다. THP 를 사용할때 디폴트로 리눅스는 가능한 모든 영역에 대하여 Huge Page 를 선택한다. khugepaged 프로세스는 백그라운드에서 각 프로그램의 가상 주소 공간을 스캔하고, 가능한 경우 스캔된 normal page 들을 huge page 로 승격시킨다. 이 때, khugepaged 스캔에 요구되는 오버헤드가 시스템의 성능 저하를 유발할 수 있으므로 시스템 관리자는 khugepaged 프로세스를 얼마나 공격적으로 실행할지를 구성할 수 있다. Huge Page 할당을 위해 메모리 예약을 사용하는 hugetlbfs 와 달리 THP 는 각 승격마다 한 개의 Huge Page 에 대한 연속적인 메모리를 buddy allocator 시스템에게 할당한다. 프로그램은 큰 노력없이 THP 를 사용할 수 있으므로 많은 리눅스 배포판에서 THP 가 디폴트로 설정되어 있다.

그럼에도 불구하고 THP 에는 여러 제약 사항들이 있다. 첫 번째 문제는 memory bloating 이다. memory bloating 이란 많은 오브젝트들의 할당으로 인해 메모리 사용량이 급격히 증가하는 것을 말한다. THP 는 normal page 보다 더 큰 크기의 페이지를 사용하기 때문에 메모리 낭비를 일으키는 메모리 내무 조각화가 발생할 수 있다.[1] Redis, MongoDB 를 포함해 메모리 내무

조각화를 겪는 프로그램들은 사용자에게 THP 를 끌 것을 권장하고 있다.([11], [12]) 또 다른 문제점은 연속적인 메모리 할당의 실패이다. THP 가 Huge Page 를 위한 연속적인 메모리 할당에 실패하면 normal page 를 사용하는 것으로 fall back 한다. 그러므로 Huge Page 할당이 빈번하게 실패하게 되면 THP 의 이점이 줄어들게 된다. 실제로 초기 버전의 리눅스에서는 Huge Page 할당에 실패하였을 때, 메모리 압축을 한 후 THP 의 이점을 유지하기 위해 Huge Page 의 재할당을 시도했다. 하지만 메모리 압축은 Huge Page 할당의 latency 를 크게 증가시키기 때문에 최신 버전의 리눅스는 Huge Page 할당 실패 시 normal page 사용으로 fall back 하도록 수정했다. [13]

요약하면 Huge Page 는 유용하고, THP 가 Huge Page 의 구현에 효과적으로 기여할 수 있다. 하지만 많은 오브젝트들의 할당으로 인한 메모리 사용량의 급격한 증가와 빈번한 Huge Page 할당이 실패하는 상황에서는 THP 의 이점이 줄어들며 이는 기존의 시스템에서 일반적이다. 본 연구에서도 리눅스의 THP 를 사용하였다.

## 2.3 Memory Access Tracing Technique

memory access tracing 기술은 메모리 접근에 대한 다양한 정보를 제공한다. 메모리 추적 프로그램은 메모리 영역에 대한 hotness 와 coldness 정보를 제공하여 중요한 자원인 메모리를 효율적으로 관리할 수 있게 한다. 이러한 memory access tracing 기술은 크게 두 가지로 분류할 수 있다.

첫 번째는 access bit 추적 방법이다. [14] access bit 를 이용한 추적 방식은 메모리 접근 모니터링에 대표적으로 사용되는 기술이다. [15] 매핑된 페이지가 접근되었는지의 여부를 표시하는 페이지 테이블 엔트리에서의 access bit 를 이용한다. acess bit 는

TLB 워크나 페이지 테이블 워크에서 주소 변환에 해당 매핑이 사용될 때 표시된다. 소프트웨어는 주기적으로 access bit 를 clear 하여 메모리 접근을 추적할 수 있다. 이 기술은 접근 유형이나 접근된 메모리 주소와 같은 세분화된 정보를 수집하지는 않는다. 페이지 접근의 여부만을 추적한다. 프로그램의 실행을 직접적으로 제한하지 않기 때문에 오버헤드는 상대적으로 적은 기술이다. 하지만 프로그램이 더 많은 메모리를 사용하게되면 오버헤드가 증가하고, 전체 페이지를 모니터링 하는 시간 또한 증가하여 추적의 퀄리티가 급격하게 저하될 수 있다.

두 번째 방법으로는 워크로드 계측이 있다. [16] 이 기술은 광범위한 메모리 기술로 컴파일 시간 최적화[17] 등을 사용하여 대상이 되는 워크로드에 hook 을 설치한다. 워크로드가 실행되면 메모리 참조 명령 실행 앞뒤로 hook 함수를 실행하고, hook 함수는 메모리 참조 정보를 기록한다. 이 기술은 모든 메모리 접근에 대한 충분한 정보를 제공하기 때문에 정확성 평가나 성능 최적화와 같은 다양한 분야에 활용될 수 있다. 그러나 이 기술은 모든 메모리 접근에 대해 hook 을 하기때문에 추적에 많은 시간을 필요로 한다. 또한, 모든 메모리 정보를 기록하기 위해서는 매우 큰 저장 공간이 필요하다.

본 연구에서는 access bit 기술을 활용하여 워크로드의 메모리 접근 패턴 및 특징을 파악하였고, 그 결과의 분석을 토대로 해당 워크로드에서 Huge Page 를 효율적으로 적용하기 위한 조건을 설정하였다.

# 제 3 장 워크로드 분석

여기서는 여러 워크로드에 대해 normal page를 사용할때와(orig) huge page를 사용할때의(thp) 실행 시간과 최대 메모리 사용량을 분석한다. 이를 통해 THP를 효율적으로 적용할 수 있는 최적의 워크로드를 선택하여 해당 워크로드에 대한 THP 사용의 최적화를 시도한다. (해당 워크로드에 대한 THP 사용의 최적화 내용은 다음 장에서 자세하게 설명한다.)

parsec3 워크로드 13개[18]와 splash2x 워크로드 12개[19]를 워크로드 분석에 사용했다. 워크로드를 실행한 머신은 CPU는 2-way Xeon Gold 6140 2.3GHz, 18 core이고, 메모리는 64GB이며 스토리지는 SSD 250GB이다.

## 3.1 Huge Page 사용에 따른 실행시간

13개의 parsec3 워크로드와 12개의 splash2x 워크로드들에 대하여 4KB 크기의 normal page를 사용했을 때(orig)와 리눅스의 THP를 이용하여 2MB 크기의 Huge Page를 사용했을 때(thp)의 각 워크로드의 실행 시간을 살펴보았다.

거의 모든 워크로드에서(예외: parsec3의 vips, streamcluster, facesim) normal page를 사용했을 때보다 실행 시간이 단축되었으며 최대 약 25%까지 실행 시간이 단축된 워크로드도(splash2x의 fft) 있었다. 이는 Huge Page의 사용으로 주소 변환 오버헤드와 TLB miss가 감소하여 더 좋은 성능을 나타낸 것으로 볼 수 있다.

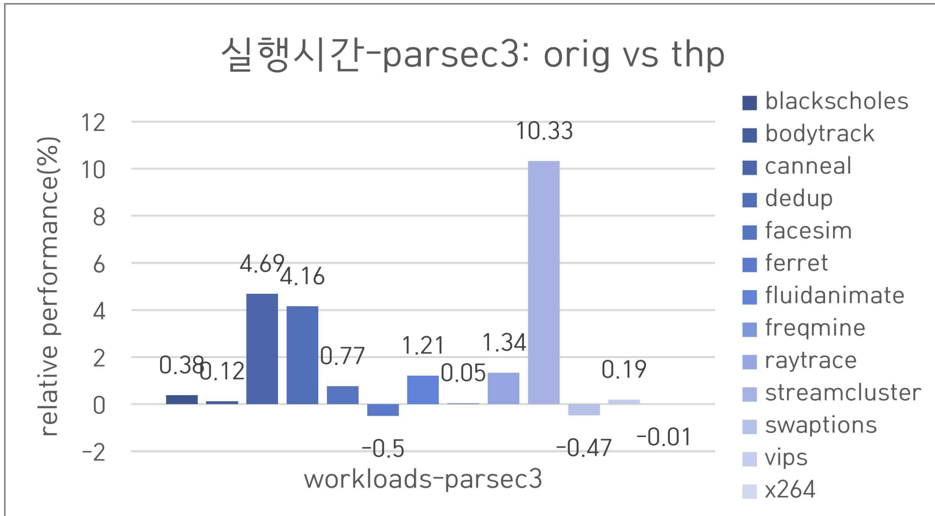



그림 3.1 parsec3 워크로드 실행시간

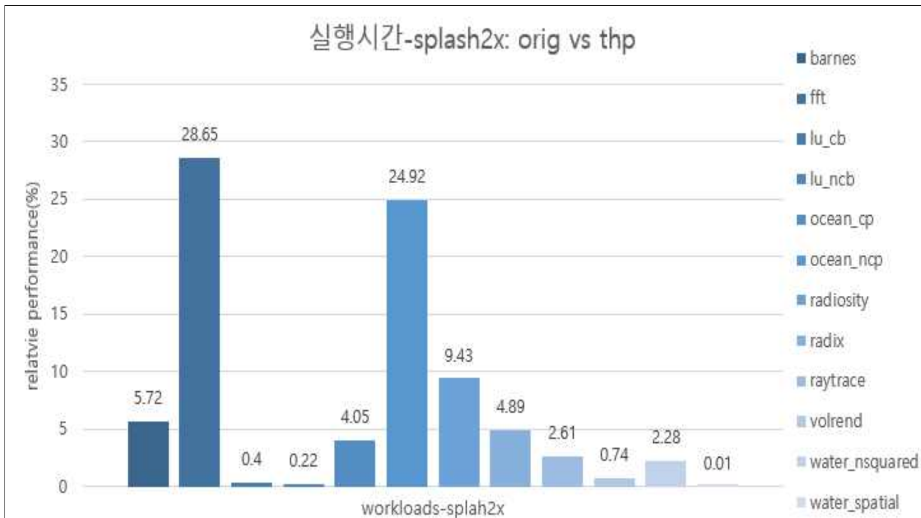



그림 3.2 splash2x 워크로드 실행시간

## 3.2 Huge Page 사용에 따른 최대 메모리 사용량

13개의 parsec3 워크로드와 12개의 splash2x 워크로드들에 대하여 4KB 크기의 normal page를 사용했을 때(orig)와 리눅스의 THP를 이용하여 2MB 크기의 Huge Page를 사용했을 때(thp)의 각 워크로드의 실행 시간 동안에 최대 메모리 사용량을 측정해 살펴보았다.

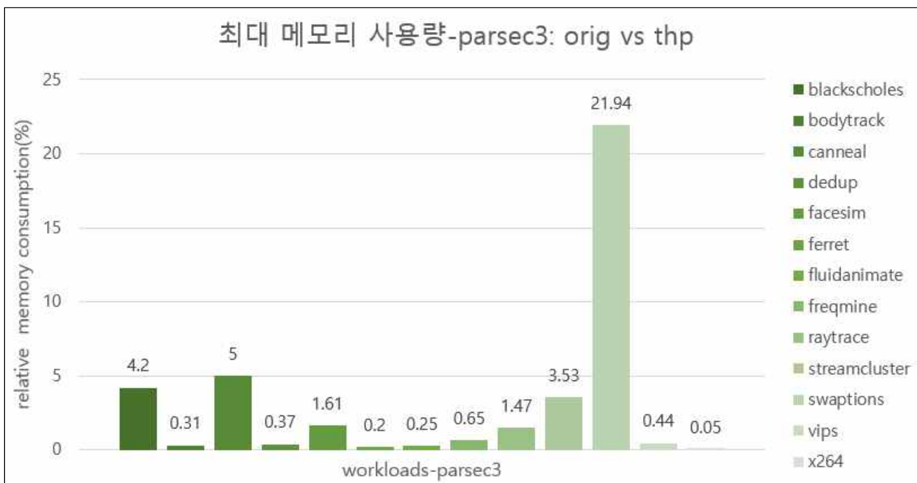


그림 3.3 parsec3 워크로드 최대 메모리 사용량

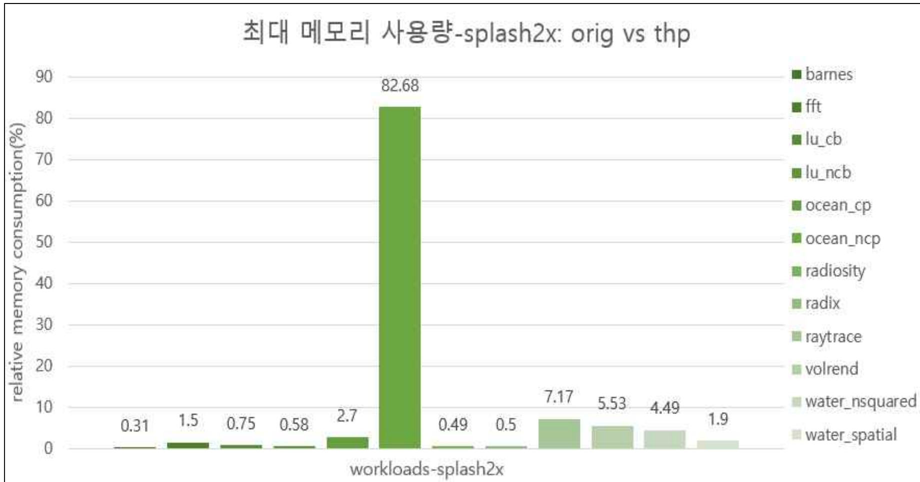



그림 3.4 splash2x 워크로드 최대 메모리 사용량

Huge Page를 사용했을 때 대부분의 워크로드가 수행 시간은 단축되었던 반면, 최대 메모리 사용량은 normal page를 사용했을 때 보다 모두 증가한 것을 볼 수 있다. 최대 82.68%까지 메모리 사용량이 증가한 워크로드도(splash2x의 ocean_ncp) 있었다. 이는 normal page(4KB) 보다 512배 더 큰 크기의 Huge Page(2MB)를 사용하기 때문에 한 개의 페이지에 적재되는 메모리 영역이 증가하므로 최대 메모리 사용량이 증가한 것으로 볼 수 있다.

## 3.3 워크로드 분석 결론 및 최적의 워크로드 선택

워크로드의 실행시간과 최대 메모리 사용량을 함께 고려해보았을 때 Huge Page의 사용을 효율적으로 최적화하기에 알맞은 워크로드를 선택한다. 그러므로 normal page를 사용할 때보다 huge page를 사용할 때 실행 시간은 5% 이상 단축되면서 최대 메모리 사용량은 5% 이상 증가하는 워크로드를 선택한다. 이 조건을 모드 만족하는 워크로드는 splash2x의 ocean_ncp이다. 해당 워크로드에 대해 실행 시간은 Huge Page를 사용할때에 근접하게 유지하면서

최대 메모리 사용량은 normal page를 사용할때보다 크게 증가하지 않게 Huge Page의 사용을 최적화하는 작업을 진행한다.

# 제 4 장 설계 및 구현

4 장에서는 3 장에서의 분석 결과로 선택된 splash2x 의 ocean_ncp 워크로드를 가지고 Huge Page 를 보다 효과적으로 사용하기 위한 최적화 과정을 설명한다. 이를 위해 memory access tracing 도구인 daptrace[20]를 활용하여 ocean_ncp 의 메모리 사용 패턴 특징을 분석한다. 이를 바탕으로 Huge Page 를 사용했을 때의 이점을 최대한 얻을 수 있는 부분만을 선택하여 해당하는 메모리 영역만 Huge Page 를 사용하고 나머지 부분은 normal page 를 사용한다. 이를 통해 Huge Page 를 사용했을 때 성능상의 이점은 누리면서 메모리 내부 조각화로 인해 낭비되는 메모리양은 최소화하여 Huge Page 사용을 최적화한다. Huge Page 의 사용은 리눅스에서 지원하는 THP; Transparent Huge Page 를 이용했다.

## 4.1 Memory Access pattern 확인

Huge Page 의 사용을 최적화 하기 위해 ocean_ncp 워크로드의 수행 시간 동안 메모리 접근 패턴을 상세하게 파악하고, 특징을 살펴보았다. 메모리 접근 패턴을 자세하게 분석하고 특징을 파악하기 위해 memory access tracing 도구인 daptrace 를 활용한다. daptrace 는 access bit 추적 기술을 기반으로 하는 memory access tracing 도구이다. daptrace 는 낮은 tracing 오버헤드를 위해 샘플링의 방식을 이용한다. 수행되는 워크로드의 메모리 접근을 기록할 때 접근 빈도수가 유사하여(접근 빈도수가 10% 이내 차이) 하나의 영역으로 지정된 영역 내에 속하는 임의의 페이지를 하나 선택하여 그 페이지의 access bit 을 확인한다. 만약 해당 페이지가 접근되었다면 그 페이지가 속한 영역 전체가 접근된 것으로 간주한다. 하지만 본 연구에서는 정확성을 높이기 위해

샘플링의 방식을 사용하지 않고, ocean_ncp 프로그램의 전체 가상 주소 공간을 4KB 단위로 나누어 모니터링 주기마다 각 영역에 속한 모든 페이지들의 access bit 를 확인하여 해당 영역의 메모리 접근을 카운팅하여 기록하도록 수정하여 사용했다.

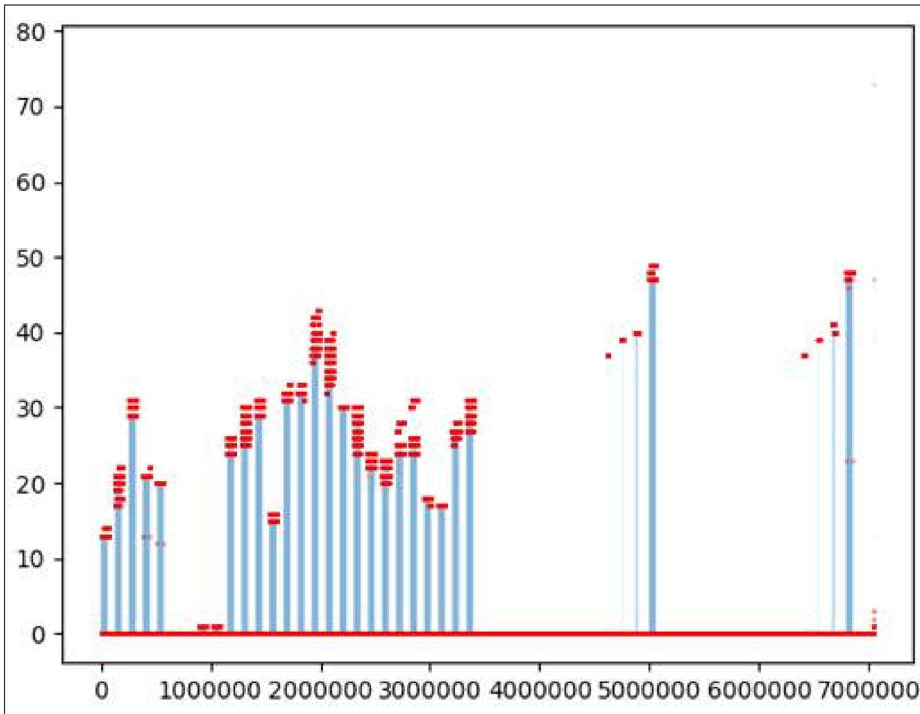



그림 4.1 ocean_ncp 의 메모리 접근 패턴

daptrace 를 활용한 메모리 접근 패턴 분석을 통해 그림 4.1 의 결과를 얻을 수 있었다. 그림 4.1 에서 y 축은 ocean_ncp 가 수행되는 동안 해당 메모리 영역의 접근 빈도수를 나타내며 x 축은 ocean_ncp 워크로드의 가상 메모리 페이지 주소의 인덱스를 나타낸다. ocean_ncp 의 가상 주소 범위는 약 27GB 이고, 수행 시간은 약 120 초이다. 이 결과를 통해 ocean_ncp 메모리 접근 패턴의 특징을 분석하였다. 그 결과 워크로드를 수행하는 동안 유사한 패턴을 보이며 반복적으로 접근되는 메모리 영역이 있음을 알 수 있었다.

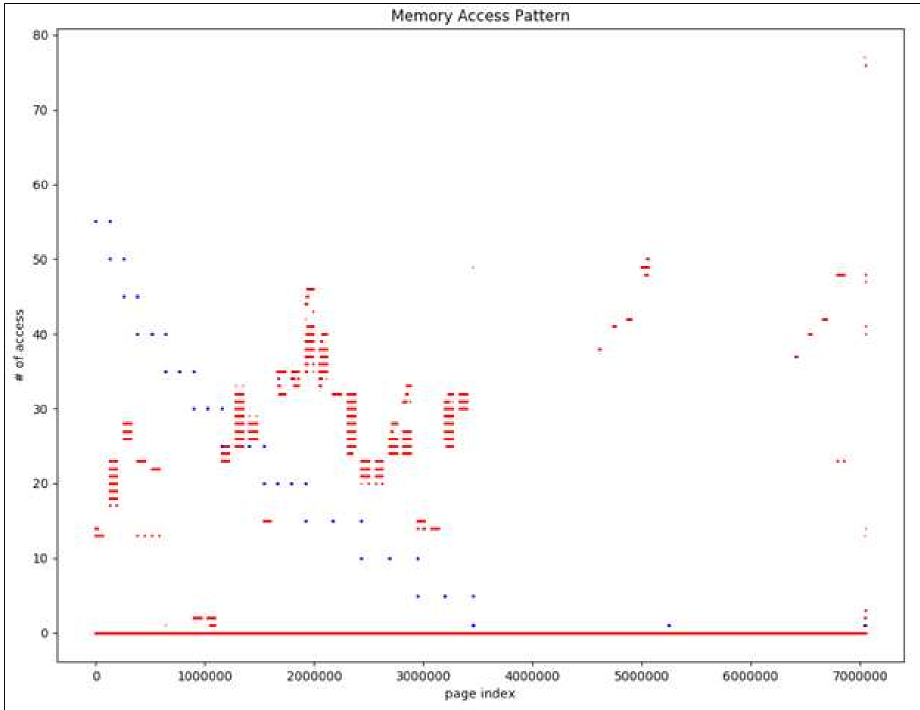



그림 4.2 ocean_ncp 에서 반복적으로 접근되는 메모리 영역

## 4.2 효율적인 Huge Page 사용을 위한 최적화 조건 탐색

ocean_ncp 에서 4.1 에서의 메모리 접근 패턴 분석을 토대로 워크로드 수행시 유사한 패턴을 가지면서 반복적으로 접근되는 메모리 영역에 해당하는 .자료 구조를 조사했다. 해당 자료 구조는 ocean_ncp 에서 500MB 이상의 크기를 가지며 malloc 으로 할당되는 구조체 변수들임을 알 수 있었다. (그림 4.2, 4.3)

또한, 그림 4.1 에서 0 번부터 1024 번까지의 페이지 인덱스 부분을 확대한 그림 4.4 를 통해서는 normal page(4KB)로 워크로드르 수행할때보다 Huge Page(2MB)로 수행할 때 약 2 배 이상의 메모리 낭비가 야기됨을 알 수 있다.

| SIZE(GB) | NAME |
|---|---|
| 13.68 | multi |
| 1.95 | wrk4, wrk5, fields |
| 1.47 | wrk1, wrk3 |
| 0.98 | iter, fields2, guess |
| 0.49 | wrk2, wrk6, frcng |

그림 4.3 ocean_ncp 의 자료구조

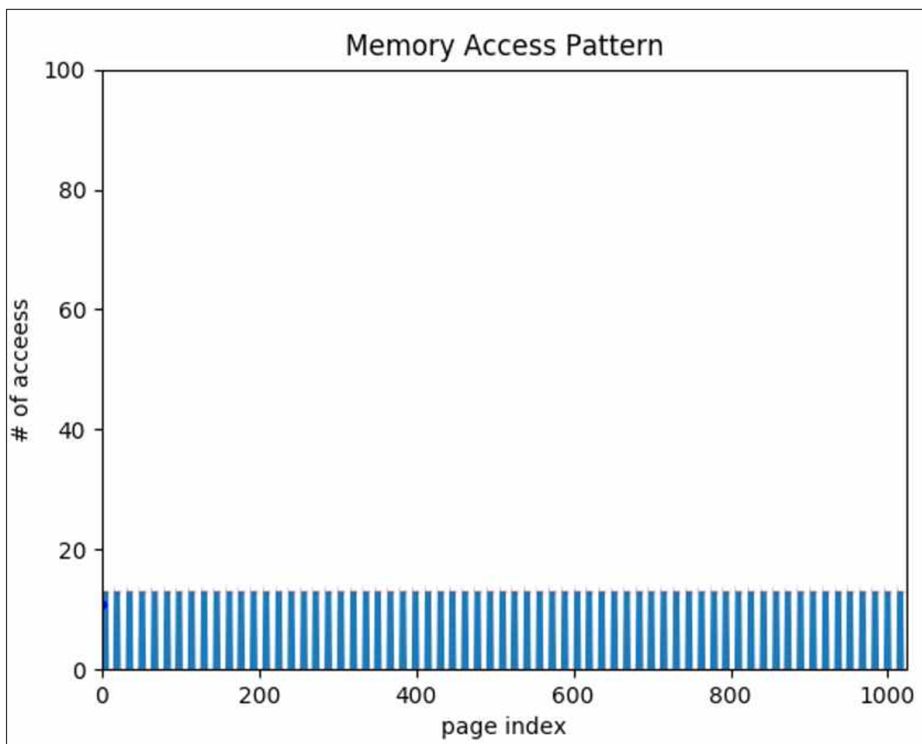



그림 4.4 0 번부터 1024 번 페이지까지의 메모리 접근 패턴

기존처럼 ocean_ncp 프로그램 전체 메모리 영역에 대하여 모두 Huge Page를 적용하는 것보다 유사한 패턴을 가지면서 반복적으로 접근되는 500MB 이상의 메모리 영역을 차지하는 오브젝트에 대해서만 한정하여 Huge Page를 사용하는 것이 더욱 효율적일 수 있음을 알아냈다. 따라서 Huge Page 한 개보다 큰 크기를 가지면서 반복적으로 접근되는 오브젝트들에 대해서만 Huge Page를 사용한다는 조건으로 Huge Page를 적용한다면 Huge Page로 인해 얻는 성능상의 이점은 얻으면서 최대 메모리 사용량의 증가량은 감소할 것이라고 분석하였다.

## 4.3 선택적인 Huge Page 적용 MADV_HUGEPA

원하는 오브젝트가 차지하는 메모리 영역에 대해서만 선택적으로 Huge Page를 적용하기 위해 madvise 시스템 콜과 MADV_HUGEPAGE 플래그를 사용했다. [21]

madvise 시스템 콜과 MADV_HUGEPAGE 플래그를 통해 사용자가 지정한 시작 주소와 크기 만큼에 해당하는 범위에 있는 페이지들에 대해 THP를 활성화한다. 이 기능은 리눅스 2.6.38 이후로 제공되는 기능이다. 커널은 정기적으로 Huge Page 후보들로 표시된 영역들을 스캔하여 Huge Page로 승격시킨다. 또한, posix_memalign 함수를 이용하여 해당 영역이 Huge Page 크기에 맞춰 정렬되어질 때 곧바로 해당 영역을 Huge Page로 승격시킨다. 본 연구에서는 선택적으로 Huge Page를 사용할 영역들에 대해 posix_memalign 함수를 사용하여 정렬함으로써 곧바로 해당 영역이 Huge Page로 승격될 수 있게 하였다. 이 기능은 주로 큰 크기의 데이터와 그에 해당하는 큰 크기의 메모리 영역을 한번에 접근하는 응용프로그램을 대상으로 사용된다. (예를 들어, QEMU와 같은 가상화 시스템) 하지만 이 기능을 사용했을 때 매우 심각한 메모리

낭비가 발생할 수도 있다. 예를 들어, 1 바이트만을 접근하는 영역에 대해 4KB 크기의 normal page 대신 2MB 크기의 Huge Page를 사용하는 경우 등이 발생할 수 있다. 그러므로 사용자가 미리 알고 있는 메모리 접근 패턴 영역에 대해 이 기능을 사용해야 응용 프로그램의 메모리 풋 프린트가 급격하게 증가할 위험성을 방지할 수 있다.

본 연구에서는 ocean_ncp에서 500MB 이상의 크기를 갖는 12개의 구조체 변수들에 대해 각각의 구조체가 할당되는 영역에 대해 madvise 시스템 콜과 MADV_HUGEPAGE 플래그를 사용하여 선택적으로 Huge Page의 사용을 적용했다.

그림 4.5 Huge Page의 선택적 사용

# 제 5 장 실험 및 분석

하나의 머신을 사용해 실험한다. 머신은 CPU는 2-way Xeon Gold 6140 2.3GHz, 18 core이고, 메모리는 64GB이며 스토리지는 SSD 250GB이다. 실험에서는 워크로드 분석을 통해 선택한 splash2x의 ocean_ncp의 데이터 구조 중에서 500MB 이상의 크기를 가지는 구조체 타입의 변수 12개에 대하여 가장 큰 크기를 갖는 변수부터 단계적으로 해당 메모리 영역에 대해서만 Huge Page를 사용하게 한다.

위와 같은 방식으로 선택적으로 특정 메모리 영역에 대해서만 Huge Page를 사용했을 때 각 단계마다 ocean_ncp의 수행 시간과 최대 메모리 사용량을 비교한다.

orig는 ocean_ncp 전체에 대해 normal page를 사용한 경우, thp는 ocean_ncp 전체에 대해 Huge Page를 사용한 경우를 나타낸다. lv0부터 lv4_2까지는 500MB 이상의 크기를 갖는 구조체 변수들에 대하여 크기가 가장 큰 순서부터 단계적으로 해당 구조체

변수가 할당되는 메모리 영역에 대해서만 부분적으로 thp 를 활성화시킨다.

## 5.1 ocean_ncp 실행시간

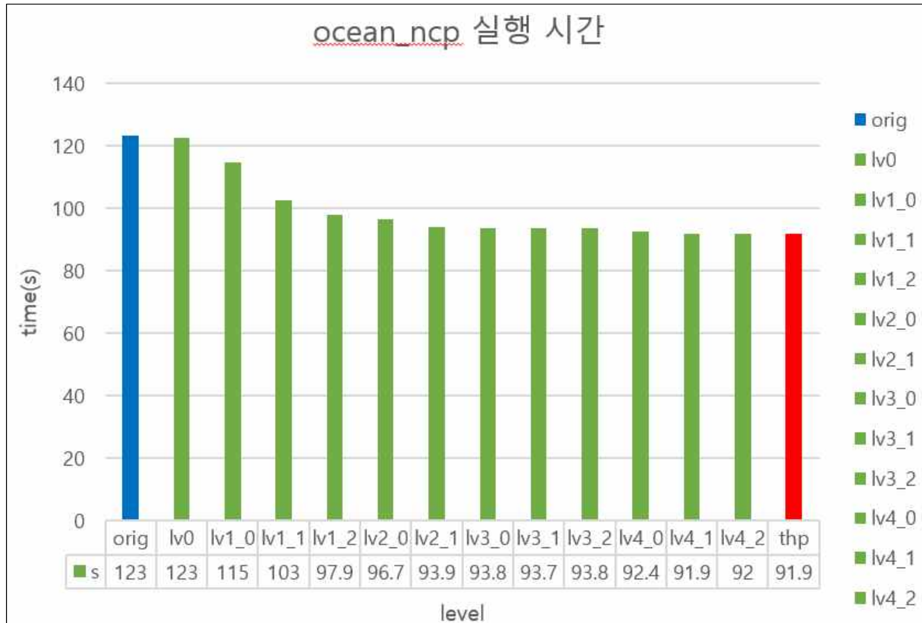



그림 5.1 단계별 Huge Page 선택적 사용에 따른 실행시간

그림 4.6 에서는 단계별로 선택적인 Huge Page 를 적용할 때마다 ocean_ncp 전체에서 normal page 를 사용하는 orig 의 경우보다 실행 시간이 단축됨을 볼 수 있다. 특히, lv1_0 에서 lv1_1 로 이행하는 구간에서 가장 큰 폭으로 실행 시간이 단축되었다.(약 10%의 실행 시간 단축) 또한, 단계를 진행해 나갈수록 ocean_ncp 전체에 Huge Page 를 사용하는 thp 와 거의 비슷한 수준으로 실행 시간이 단축됨을 알 수 있다. 하지만 lv2_1 부터는 실행 시간 단축의 정도가 거의 차이나지 않는다는 것을 볼 수 있다.

## 5.2 ocean_ncp 최대 메모리 사용량

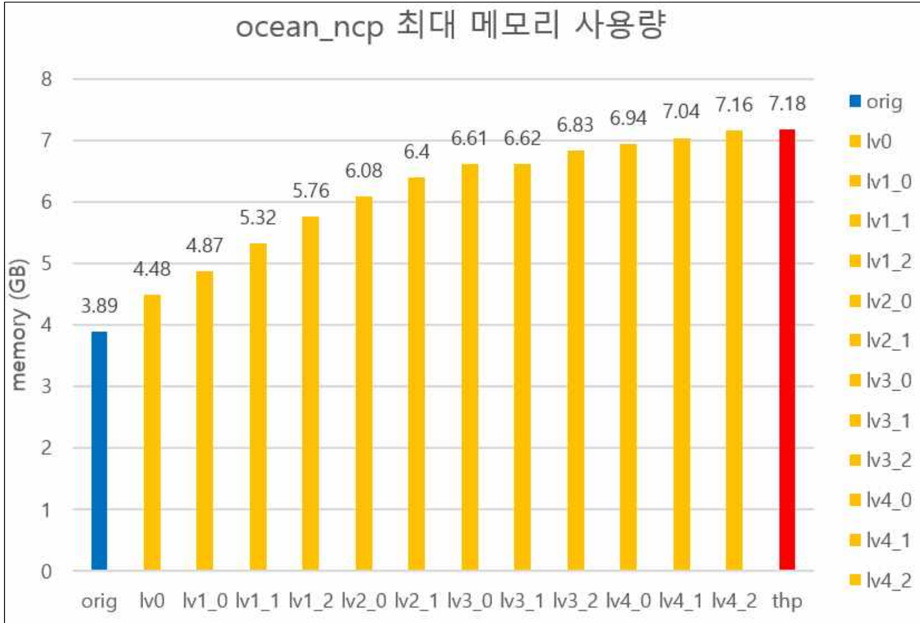



그림 5.2 단계별 Huge Page 선택적 사용에 따른 최대 메모리 사용량

그림 4.7 에서는 단계별로 선택적인 Huge Page 를 적용할 때마다 ocean_ncp 전체에서 normal page 를 사용하는 orig 의 경우보다 최대 메모리 사용량이 증가함을 볼 수 있다. 단계에 따른 메모리 사용량은 linear 하게 증가하는 경향을 보인다. 특히, lv2_1 부터는 orig 의 경우와 비교했을 때 65%이상의 최대 메모리 사용량 증가율을 보였다. 이 경우 워크로드가 사용할 수 있는 메모리가 제한적인 상황이라면 메모리 풋 프린트의 증가로 인한 심각한 성능저하가 유발될 수 있다. 단계를 진행해나갈수록 ocean_ncp 전체에 Huge Page 를 사용하는 thp 와 거의 비슷한 수준까지 최대 메모리 사용량이 증가함을 볼 수 있다. 이를 통해 해당 워크로드 수행시 500MB 이상의 크기를 가지는 12 개의 구조체 변수들이 메모리 접근의 대부분을 차지하고있으며 프로그램 전체 메모리 사용량의 주요 요소임을 확인할 수 있다.

그림 4.6 과 그림 4.7 의 결과를 통해 madvise 시스템 콜과 MADV_HUGEPAGE 를 사용하여 단계적으로 특정 메모리 영역에 대해서만 Huge Page 를 적용할 때에도 실행 시간 단축과 최대 메모리 사용량은 서로 trade-off 임을 알 수 있다. 실행 시간의 경우 특정 단계 이상에서는 유의미하게 실행 시간 단축이 이루어지지않음을 알 수 있었다. 반면, 최대 메모리 사용량의 경우에는 단계별로 진행해나감에 따라 linear 하게 증가함을 알 수 있었다. 그러므로 각 단계마다 실행시간의 단축과 최대 메모리 사용량의 증가의 trade-off 를 고려하여 선택적 Huge Page 사용을 적용해야 한다.

# 제 6 장 관련연구

Huge Page를 보다 효율적으로 사용하기 위한 연구로는 메모리 페이지의 사용율과 memory access pattern을 추적함으로써 메모리 연속성을 자원으로 관리하여 tranparent 한 Huge Page 지원을 제공하는 프레임 워크를 설계한 연구가 있다. [1]. 또 다른 연구로는 보장된 연속 메모리 할당자를 통해 효율적인 메모리 활용과 짧은 대기 시간, 성공적인 메모리 할당을 보장함으로써 Huge Page 사용의 효율성을 증가시킨 것이 있다. [10] 두 연구 모두 Huge Page 사용 시 낭비되는 메모리 양에 대한 고려보다는 연속적인 메모리 영역을 자원으로 잘 관리하여 Huge Page 할당이 최대로 가능하게하여 Huge Page 사용의 성능을 증가시키는데 초점을 두고 있다.

# 제 7 장 결론

본 논문에서는 오늘날 테라바이트 이상의 메모리를 지원하는 최신의 컴퓨팅 플랫폼과 이러한 대용량 메모리를 활용하는 워크로드들이 일반화되어 대용량 메모리를 사용하는데 있어 TLB miss 와 주소 변환 오버헤드로 인해 이를 해결하기 위한 Huge Page 사용에 대해 분석하고, 보다 효율적으로 워크로드에 Huge Page 를 적용할 수 있는 방법을 탐색하였다. Huge Page 를 사용했을때의 실행시간 단축과 최대 메모리 사용량 증가의 분석을 통해 최적화된 Huge Page 를 적용하기 위한 워크로드를 선택하고, memory aceess tracing 도구를 사용하여 해당 워크로드의 메모리 접근 패턴을 상세히 분석하였다. 메모리 접근 패턴을 통해 크기가 500MB 이상인 데이터 구조에 대해 선택적으로 Huge Page 를 적용하는 방식으로 그때의 실행시간 감소와 최대 메모리 사용량 증가를 비교하였다.

단계적으로 선택적인 Huge Page 적용을 수행함으로써 각 단계별로 실행시간 감소와 최대 메모리 사용량 증가를 비교하여 적절한 trade-off 를 갖는 단계를 선택하여 Huge Page 의 효율적인 사용을 할 수 있다.

# 참고문헌


[1] Y. Kwon, H. Yu, S. Peter, C. J. Rossbach, and E. Witchel, “Coordinated and efficient huge page management with ingens,” in Proc. 12th USENIX Symp. Operating Syst. Des. Implementation, 2016, pp. 705 – 721.

[2] Intel Corporation. Intel-64 and IA-32 Architectures Software Developers Manual 2016.

https://www-ssl.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-softwaredeveloper-manual-325462.pdf.

[3] http://www.7-cpu.com/cpu/Skylake.html. [Accessed April, 2016].

[4] SAP IQ recommends disabling huge pages. http://scn.sap.com/people/markmumy/blog/2014/05/22/sap-iq-and-linuxhugepagestransparent-hugepages. [May, 2014].

[5] Splunk recommends disabling huge pages. http://docs.splunk.com/Documentation/Splunk/6.1.3/ReleaseNotes/SplunkandTHP. [December, 2013].

[6] Redis recommends disabling huge pages. http://redis.io/topics/latency. [Accessed April, 2016].

[7] MongoDB recommemds disabling huge pages. https://docs.mongodb.org/manual/tutorial/transparent-huge-pages/. [Accessed April, 2016].

[8] CouchBase recommends disabling huge pages. http://blog.couchbase.com/oftenoverlooked-linux-os-tweaks. [March, 2014].

[9] http://www.7-cpu.com/cpu/Haswell. html. [Accessed April, 2016].

[10] GCMA: Guaranteed Contiguous Memory Allocator, 2019

[11] “Redis Latency Problems Troubleshooting.” [Online]. https://redis.io/topics/latency

[12] “Disable Transparent Huge Pages (THP) - MongoDB Manual 3.0.” [Online]. https://docs.mongodb.com/v3.0/tutorial/transparent-huge-pages/

[13] V. Babka, “[PATCH v5 6/8] mm, thp: remove __gfp_noretry from khugepaged and madvised allocations.” (2016). [Online]. https://marc.info/?l=linux-kernel&m=146908669009067&w=2

[14] SeongJae Park, Yunjae Lee, Heon Y. Yeom, “"Profiling Dynamic Data Access Patterns with Controlled Overhead and Quality", Middleware ‘19

[15] Waldspurger, C., Saemundsson, T., Ahmad, I., Park, N.: Cache modeling and optimization using miniature simulations. In: 2017 USENIX Annual Technical Conference (ATC). USENIX Association, Santa Clara, CA, pp. 487 - 498 (2017)

[16] Yunjae Lee, Yoonhee Kim2, Heon Y. Yeom, "Lightweight memory tracing for hot data identification", Cluster Computing (2020) 23:2273 – 2285 https://doi.org/10.1007/s10586-020-03130-1

[17] Chang, P.P., Mahlke, S.A., Hwu, W.M.W.: Using profile information to assist classic code optimizations. Software 21(12), 1301 – 1321 (1991)

[18]http://lacasa.uah.edu/portal/Upload/tutorials/parsec/Running_Parsec-3.0.pdf

[19] https://parsec.cs.princeton.edu/parsec3-doc.htm#splash2x

[20] http://dcslab.snu.ac.kr:30480/daptrace/daptrace

[21] https://www.man7.org/linux/man-pages/man2/madvise.2.html

# ABSTRACT

Recently, platforms supporting terabytes of more of memory and workloads utilizing large amounts of memory are becoming commonplace. The use of large memory has limitations in address translation overhead and TLB miss. As a one of solutions, you can use Huge Pages. However, in most cases it is recommended to disable Huge Pages on the system despite the clear performance advantages due to memory watste caused by Huge Pages. Therefore, in this paper, through analysis of execution time and maximum memory usage, when using Huge Pages in various workloads, a workload that can apply Huge Pages more efficiently was selected. Then, using a memory access tracing tool, we analyzed the memory access pattern of the workload to explore how to use Huge Pages efficiently. By applying a Huge Page only to a memory area corresponding to a data structure having a size of 500MB or more that is repeatedly accessed, a huge page is optimally applied in the workload by determining an appropriate trade-off between execution time and maximum memory usage.