계층형 메모리 시스템을 위한 동적 페이지 마이그레이션 기법

(Dynamic Page Migration Technique for Tiered Memory Systems)

이성민*, 신지우*, 정진만**

(Seongmin Lee, Jiwoo Shin, Jinman Jung)

요 약

계층형 메모리 시스템은 제한된 용량의 빠른 계층과 대용량의 느린 계층을 결합하여 성능과 비용을 최적화하는 구조로, 최근 메모리 집약적 워크로드에서 효과적인 수단으로 인식된다. 이러한 계층형 메모리 시스템에서 페이지 마이그레이션은 핵심 수단이지만 워킹셋 크기(Working Set Size, WSS)가 빠른 계층 메모리의 용량을 초과하게 되면 빈번한 승격과 강등으로 인한 핑퐁(ping-pong) 효과가 발생하여 메모리 대역폭이 떨어지게 된다. 본 논문은 메모리 대역폭을 이용하여 WSS의 변화를 감지하고, NUMA(Non-Uniform Memory Access) 스캔 주기를 동적으로 조절하는 페이지 마이그레이션 기법을 제안한다. 큰 워킹셋으로 판단되는 경우 NUMA 스캔 주기를 늘려 페이지 마이그레이션을 줄이고, 작은 워킹셋으로 판단되는 경우 NUMA 스캔 주기를 단축하여 페이지 마이그레이션이 적극적으로 일어날 수 있도록 한다. WSS가 동적으로 변하는 환경에서 CXL(Compute Express Link) 기반 계층형 메모리 시스템을 모사하여 실험한 결과 제안 기법은 최신 메모리 관리 기법인 TPP 대비 최대 21.12%의 성능 향상을 달성하였다.

■ 중심어 : 계층형 메모리 시스템 ; CXL ; 페이지 마이그레이션 ; 메모리 대역폭

Abstract

Tiered memory systems, which combine a small, fast tier with a large, slow tier, are increasingly recognized as an effective approach to optimizing both performance and cost in memory-intensive workloads. In such systems, page migration plays a central role; however, when the working set size (WSS) exceeds the capacity of the fast memory tier, frequent promotions and demotions can lead to a ping-pong effect that significantly degrades memory bandwidth. This paper proposes a page migration technique that dynamically adjusts the NUMA scan period by detecting changes in WSS based on memory bandwidth. When a large WSS is detected, the scan period is increased to reduce page migrations; conversely, when a small WSS is detected, the scan period is decreased to allow more aggressive page migration. Experiments simulating a CXL-based tiered memory system under dynamically changing WSS demonstrate that the proposed technique achieves up to 21.12% performance improvement compared to the state-of-the-art memory management scheme, TPP.

■ keywords : Tiered Memory System ; CXL ; Page Migration ; Memory Bandwidth

Ⅰ. 서 론

현대 대규모 컴퓨팅 시스템에서는 인공지능(AI) 모델 학습, 대규모 그래프 분석, 실시간 데이터 처리 등 메모리 집약적 워크로드의 증가로 인해 메모리 용량 수요가 급격히 증가하고 있다.

* 정회원, 인하대학교 전기컴퓨터공학과
** 종신회원, 인하대학교 컴퓨터공학과
이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.RS-2023-00252501).
본 연구는 2025년 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학사업의 연구결과로 수행되었음(2022-0-01057).
접수일자 : 2025년 08월 13일 게재확정일 : 2025년 09월 18일
수정일자 : 2025년 09월 05일
교신저자 : 정진만 e-mail : jmjung@inha.ac.kr

예를 들어, 초거대 언어 모델의 학습과 추론 과정에서는 수백 GB에서 수 TB에 달하는 메모리 자원이 요구되며, 이는 단일 계층 DRAM 확장만으로는 용량과 비용의 제약을 동시에 해결하기 어렵다[1]. 이에 따라 제한된 용량의 빠른 계층과 대용량의 느린 계층을 결합하는 계층형 메모리 시스템이 도입되고 있다. 이러한 환경에서는 핫 데이터는 빠른 계층, 콜드 데이터는 느린 계층에 배치함으로써 성능을 극대화한다. 최근 CXL(Computer Express Link)[2] 표준의 등장으로 CPU가 외부 메모리 장치를 단일 물리 주소 공간을 통해 고속으로 접근할 수 있는 시스템 구조가 점차 확산되고 있다. 이로 인해 운영체제가 이기종 메모리 간의 성능 차이를 고려하여 페이지 배치를 직접 관리해야 할 필요성이 커졌다.

일반적인 메모리 환경에서 운영체제는 페이지 마이그레이션을 통해 메모리 배치를 동적으로 조정한다. 리눅스는 주기적으로 주소 공간을 스캔하고 힌트 폴트(hint-fault)를 유도하여 페이지 승격 여부를 결정한다[3][4]. 실제 페이지 접근 추적을 통해 메모리 접근의 정확한 정보를 얻을 수 있다는 장점이 있으나, 페이지 폴트(page fault)에 따른 오버헤드가 존재한다. 이때 스캔 주기의 설정은 페이지 승격이 얼마나 자주 발생하는지에 영향을 미친다.

특히 워킹셋 크기(Working Set Size, WSS)가 빠른 계층의 수용 용량을 초과하면 페이지의 승격과 강등이 짧은 주기로 반복되는 핑퐁(ping-pong) 효과가 발생한다. 이는 힌트 폴트 처리 및 PTE 갱신 등 관리 오버헤드를 증가시킨다. 결과적으로 애플리케이션 관점에서의 메모리 대역폭이 감소하며, 심한 경우 마이그레이션을 수행하지 않는 상태보다 낮은 성능을 보이게 된다. 따라서 이러한 핑퐁 효과가 발생하는 시점을 조기에 식별하여 불필요한 마이그레이션을 억제하는 제어가 요구된다.

본 논문에서는 워킹셋 크기에 따라 스캔 주기를 동적으로 조절하여 큰 워킹셋에서는 불필요한 마이그레이션을 줄이고, 작은 워킹셋에서는 빠르게 승격을 유도할 수 있는 페이지 마이그레이션 기법을 제안한다. 제안 기법은 메모리 대역폭을 이용하여 워킹셋 크기의 변화를 감지하고, 시스템 상태를 적극적 스캔(Aggressive Scan)과 보수적 스캔(Conservative Scan)으로 구분한다. 작은 워킹셋 크기를 가지는 상태에서는 스캔 주기를 단축하여 페이지 마이그레이션의 반응성을 높이고, 큰 워킹셋 크기를 가지는 상태에서는 스캔 주기를 증가시켜 불필요한 페이지 마이그레이션 오버헤드를 최소화한다.

논문의 구성은 다음과 같다. 2장에서는 페이지 관리 기법 관련 연구를 살펴본다. 3장에서는 이러한 분석을 바탕으로 제안하는 동적 페이지 마이그레이션 기법의 시스템 모델을 제시하고, 시스템 상태 갱신과 스캔 주기 조절 알고리즘을 상세히 기술한다. 4장에서는 제안 기법의 성능을 평가하고 기존 기법과의 비교를 통해 효율성을 검증한다. 마지막으로 5장에서는 연구 결과를 요약하고 향후 연구 방향과 결론을 제시한다.

Ⅱ. 관련 연구

계층형 메모리 시스템에서는 서로 다른 성능 특성을 가지는 메모리 계층 간에 데이터를 효율적으로 배치하는 것이 성능 향상에 중요한 역할을 하며, 이를 위해 페이지 마이그레이션 기법이 활용된다. 페이지 마이그레이션은 일반적으로 핫 데이터를 빠른 계층으로 승격시키고 콜드 데이터를 느린 계층으로 강등하는 전략을 기반으로 한다. 이 장에서는 페이지 마이그레이션 기법들을 하드웨어 기반과 소프트웨어 기반 관점에서 분류하고 각각의 특성과 한계를 살펴본다.

1. 하드웨어 기반 페이지 관리 기법

하드웨어 기반 기법은 성능 카운터를 이용하는 방법이 대표적이다[5][6]. 성능 카운터는 CPU 및 메모리에서 발생하는 다양한 이벤트를 실시간으로 수집할 수 있는 하드웨어 기능으로, 커널 또

는 사용자 공간에서 접근이 가능하며 페이지 마이그레이션 정책의 입력으로 활용된다. 이러한 방식은 운영체제의 개입을 최소화하면서도 낮은 오버헤드로 메모리 접근 패턴을 추적할 수 있다는 장점을 갖는다. 그러나 하드웨어 기반 탐지 기법은 메모리 접근 이벤트를 직접적으로 측정하기보다는 간접적으로 추론하는 방식이기 때문에 워킹셋의 변화에 민감하게 반응하지 못한다. 페이지가 급격히 핫해지는 경우에도 일정 수준의 샘플이 수집되기 전까지는 승격이 지연되며 이는 성능 저하로 이어질 수 있다.

이와 같은 한계로 인해 하드웨어 성능 카운터 기반 마이그레이션 정책은 워킹셋의 동적 변화에 대한 민감도와 적응력이 부족할 수 있다. 이에 본 논문은 성능 카운터 기반 탐지 정보를 활용하되, 시스템의 실시간 상태 정보를 함께 반영하여 NUMA(Non-Uniform Memory Access) 스캔 주기를 동적으로 조절함으로써 탐지 지연으로 인한 부작용을 완화하고 전체 시스템의 성능을 보다 안정적으로 유지할 수 있는 기법을 제안한다.

2. 소프트웨어 기반 페이지 관리 기법

대표적인 소프트웨어 기반 기법인 TPP는 페이지 승격을 위해 사용자 페이지에 대한 접근을 불가능하게 설정하여 NUMA 힌트 폴트(마이너 페이지 폴트)를 유도하고 이를 통해 페이지의 실질적인 접근 여부를 감지하여 승격 대상으로 분류한다. NUMA 힌트 폴트 기반 방식은 비교적 높은 정확도로 핫 페이지를 식별할 수 있다는 장점이 있지만 모든 페이지에 대해 주기적으로 접근 권한을 재설정하고 마이너 페이지 폴트를 처리해야 하므로 스캔 주기가 짧아지면 시스템 오버헤드도 증가하는 단점을 가진다. 또한 워크로드의 워킹셋이 빠른 계층 메모리의 용량을 초과할 경우 계속해서 불필요한 마이그레이션이 발생하여 성능 저하를 초래할 수 있다.

Nomad는 비배타적(Non-Exclusive) 메모리 티어링 기법을 사용하여 페이지를 빠른 계층으로 승격하는 경우에도 느린 계층에 shadow copy를 유지함으로써 페이지 강등의 오버헤드를 줄였다[7]. 이러한 구조는 메모리 스래싱(thrashing) 상황에서 TPP 대비 최대 6배의 성능 향상을 보였다. 하지만 여전히 빠른 계층 메모리의 용량을 초과하는 워킹셋에 적절히 대응하지 못한다.

이러한 문제를 완화하기 위해 큰 워킹셋 탐지 시 NUMA 스캔 주기를 증가시켜 마이그레이션을 최소화하는 방식을 사용하는 연구도 존재한다[8]. 그러나 고정된 주기를 사용하기 때문에 워킹셋이 동적으로 변하는 환경에서는 민감하게 반응하지 못하는 한계가 존재한다.

본 논문에서는 메모리 대역폭을 활용하여 워킹셋 크기의 변화를 실시간으로 감지하고 NUMA 스캔 주기를 동적으로 조절하는 페이지 마이그레이션 기법을 제안한다. 하드웨어 성능 카운터를 직접 활용하여 핫 페이지를 추적하기보다는 메모리 대역폭 변화를 감지하여 시스템 상태를 모델링한다. 이를 통해 핑퐁 효과와 같은 마이그레이션 병목을 방지하고 마이그레이션이 실제로 성능 이점을 제공하는 상황에서만 NUMA 스캔 주기를 감소시킨다. 핑퐁 효과가 발생하는 경우, 제안 기법은 기존 메모리 티어링 방식보다 오버헤드를 줄일 수 있다는 점에서 의의가 있다.

Ⅲ. 동적 페이지 마이그레이션 기법

본 연구에서 제안하는 기법은 메모리 대역폭 기반의 워킹셋 크기 변화 감지를 통해 시스템 상태를 갱신하고 NUMA 스캔 주기를 동적으로 조절하는 것을 핵심으로 한다.

1. 시스템 모델

본 논문에서 제안하는 기법은 사용자 공간에서 수집한 실시간 로컬 메모리 대역폭 정보를 기반으로 커널 공간에서 NUMA 스캔 주기를 동적으로 제어하는 구조를 갖는다. 그림 1에서 보는 바와 같이 전체 시스템은 사용자 공간과 커널 공간

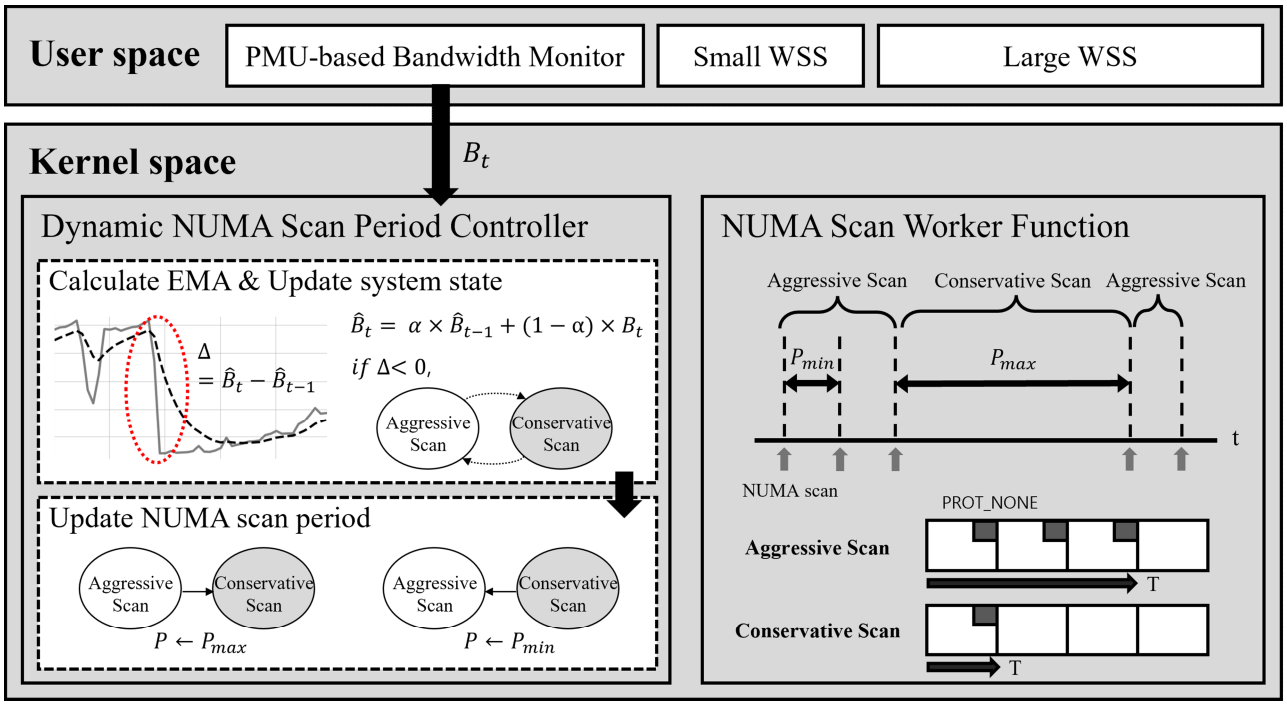


그림 1. 제안된 시스템의 개요

으로 구성된다.

사용자 공간에서는 PMU(Performance Monitoring Unit) 기반 모니터링 소프트웨어를 통해 메모리의 대역폭을 주기적으로 측정하고 해당 정보를 커널 내부의 동적 NUMA 스캔 주기 제어 스레드에 전달한다. 커널 공간에서는 전달받은 대역폭 정보를 기반으로 지수 가중 이동 평균(EMA, Exponential Moving Average)을 계산하여 시스템 상태를 실시간으로 갱신한다. 시스템 상태에 따라 NUMA 스캔 주기를 조절하여 워킹셋의 동적 변화에 반응할 수 있도록 한다.

2. 시스템 워킹셋 추정

가. EMA 기반 워킹셋 변화 추정

사용자 공간에서 PMU 기반 모니터링 소프트웨어를 통해 주기적으로 측정된 로컬 메모리 대역폭 B_t는 커널 공간으로 전달되며, 커널은 해당 값을 기반으로 수식 1을 통해 EMA 값 $\hat{B}_t$를 계산함으로써 단기적인 노이즈를 제거하고 추세를 반영한다.

$$\hat{B}_t = \alpha \cdot \hat{B}_{t-1} + (1-\alpha) \cdot B_t \quad (1)$$

로컬 메모리 대역폭을 워킹셋 변화의 추정 지표로 활용할 수 있는 근거는 다음과 같다. NUMA 환경에서 워킹셋이 로컬 메모리 내에 완전히 존재할 경우 로컬 메모리 대역폭은 높은 수준을 유지하지만, 워킹셋의 변화로 인해 페이지가 리모트 메모리로 이동하는 경우 로컬 메모리 접근량은 급격히 감소한다.

이러한 특성으로 인해 로컬 메모리 대역폭의 변화는 워킹셋의 동적 변화와 높은 상관관계를 가지며 이를 정량적으로 추적할 수 있는 효과적인 신호로 활용될 수 있다. 본 연구는 이러한 관찰을 바탕으로 로컬 대역폭을 이용해 워킹셋 변화 감지 및 NUMA 스캔 주기 조절에 활용한다.

나. 시스템 상태 갱신

시스템은 EMA 값의 변화량 Δ를 기반으로 워킹셋의 동적 변화를 탐지한다. 현재 시점의 EMA 값을 $\hat{B}_t$, 이전 시점의 값을 $\hat{B}_{t-1}$라 할 때, 두 값 간의 차이를 수식 2와 같이 정의한다.

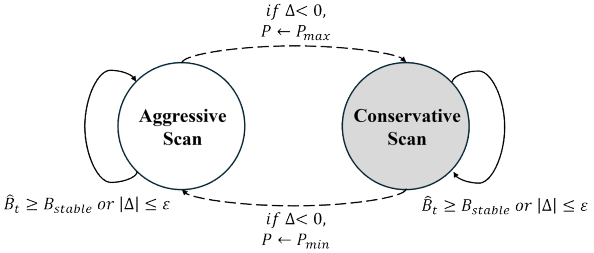


그림 2. 시스템 상태 모델

$$\Delta = \hat{B}_t - \hat{B}_{t-1} \quad (2)$$

Algorithm 1 WSS-Based System State Update

Input: Current EMA Bandwidth $\hat{B}_t$,
Previous EMA Bandwidth $\hat{B}_{t-1}$,
Global Variable: $updatePeriodFlag$,
System state S,
Stable Bandwidth Threshold B_{stable},
Variation Tolerance ε

$\Delta \leftarrow \hat{B}_t - \hat{B}_{t-1}$ // Compute the bandwidth change
if ($\hat{B}_t \geq B_{stable}$) or ($|\Delta| \leq \varepsilon$)
  $updatePeriodFlag \leftarrow False$
  return
end if
if $\Delta < 0$ // A decrease in bandwidth is observed
  if $S = Aggressive\ Scan$
    $S \leftarrow Conservative\ Scan$
  else
    $S \leftarrow Aggressive\ Scan$
  end if
end if
$updatePeriodFlag \leftarrow True$

그림 3. 시스템 상태 갱신 알고리즘

그림 2는 시스템 상태 모델을, 그림 3은 시스템 상태 갱신 알고리즘을 보여준다. $\hat{B}_t$이 사전에 정의된 안정 기준치 B_{stable} 이상이거나, 변화량 $|\Delta|$이 허용 오차 ε 이내일 경우 시스템은 현재 상태가 안정적(stable)이라고 판단한다. 이 경우 NUMA 스캔 주기 조절이 불필요하며 이후 단계는 수행하지 않는다. B_{stable}은 수식 3, ε은 수식 4로 계산할 수 있다. 이때 $B_{\max}$는 현재 하드웨어에서 실험을 통해 관측된 최대 대역폭을 나타낸다.

$$B_{stable} = \rho B_{\max},\ \varepsilon = \lambda B_{\max} \quad (3),\ (4)$$

반면 대역폭이 유의미하게 감소한 경우($\Delta < 0$) 시스템은 이를 메모리 접근 패턴의 변화로 간주하여 그림 2와 같이 시스템 상태 S를 전환한다. 현재 상태가 적극적 스캔일 경우 보수적 스캔 상태로 변경하고, 반대의 경우 다시 적극적 스캔 상태로 전환하여 NUMA 스캔 주기를 동적으로 조절한다. 이러한 방식은 워킹셋의 동적 변화에 따라 페이지 마이그레이션이 과도하게 이루어지거나 비효율적으로 발생하는 것을 방지하며 시스템 오버헤드를 최소화하면서도 워킹셋 변화에 민감하게 반응할 수 있다.

3. 스캔 주기 조절

Algorithm 2 NUMA Scan Period Update

Global Variable: $updatePeriodFlag$,
System state S,
Maximum value of the NUMA scan period P_{max},
Minimum value of the NUMA scan period P_{min}

if $updatePeriodFlag$
  return
end if
if $S = Aggressive\ Scan$
  $P \leftarrow P_{min}$
else
  $P \leftarrow P_{max}$
end if

for each process p in system do
  if (p has no memory context)
    continue
  end if
  if $p.numa\_scan\_period \neq P$ // update NUMA scan period
    $p.numa\_scan\_period \leftarrow P$
    $p.numa\_scan\_period\_max \leftarrow P$
    $p.mm.numa\_next\_scan \leftarrow jiffies + P$
  end if
end for

그림 4. 스캔 주기 조절 알고리즘

스캔 주기 조절은 시스템 상태 갱신 단계에서 워킹셋 변화가 감지되었을 경우에만 수행된다. 조절 과정은 그림 4와 같다. 현재 시스템 상태가 적극적 스캔이라면 스캔 주기를 최소 주기값 $P_{\min}$으로 설정하여 마이그레이션을 보다 적극적으로 수행하게 된다. 반면, 상태가 보수적 스캔인 경우 스캔 주기를 최대 주기값 $P_{\max}$로 설정하여 마이그레이션을 최소화시킴으로써 불필요한 마이그레이션 오버헤드를 방지한다.

이후 커널 스레드를 제외한, 시스템에 존재하는 모든 프로세스를 순회하며 각 프로세스의 NUMA 스캔 주기 관련 변수들을 갱신한다. 이를 통해 각 프로세스는 다음 스캔 시점을 새로운 주기에 맞추어 조정하게 되며 NUMA 스캔 실행 함수는 새로운 정책에 따라 동작하게 된다. 이러한 방식은 시스템 전체에 대해 통일된 마이그레이션 정책을 적용할 수 있도록 하며 NUMA 스캔 주기 변경 시 즉시 반영되도록 지원한다.

Ⅳ. 성능 평가

1. 실험 환경

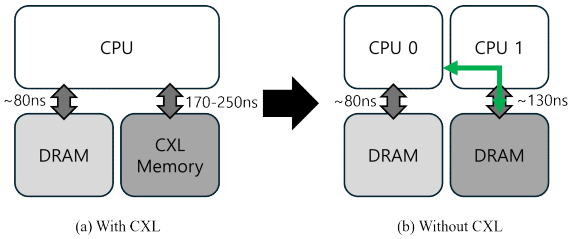


그림 5. 듀얼 소켓 NUMA 시스템을 이용한 CXL 시스템

본 논문에서는 CXL 기반 계층형 메모리 시스템을 모사하여 듀얼 소켓 NUMA 시스템을 기반으로 실험 환경을 구성하였다. 그림 5는 듀얼 소켓 NUMA 시스템을 이용한 실험 환경을 보여준다. 실험은 Ubuntu 22.04, Linux kernel v5.13.0-rc6에서 진행하였다. CXL 기반 계층형 메모리 시스템을 구성하기 위해 실험 시스템의 Socket 1의 모든 CPU 코어를 비활성화하고, 해당 소켓에 연결된 메모리를 느린 계층 메모리로 간주하였다. Socket 0의 메모리는 빠른 계층 메모리로 정의하였다. 메모리 접근 지연 시간을 Intel Memory Latency Checker(MLC) v3.11을 사용하여 측정한 결과, 빠른 계층 메모리는 80ns, 느린 계층 메모리는 130ns 수준의 지연 시간을 보였다. 이러한 수치는 실제 CXL 환경에서 관찰되는 지연 시간 차이인 2배에서 4배 수준과 유사하다[2].

Intel Xeon E5630 2-way (2.53GHz, 4-core) CPU에서 실험을 진행하였으며 빠른 계층 메모리와 느린 계층 메모리는 DDR3 8G를 2개씩, 각 16G를 가진다. 해당 실험 환경에서 $B_{\max}$ 값은 10,000으로 설정하였다.

2. 실험 결과

이 절에서는 제안하는 동적 마이그레이션 주기 조절 기법의 효과를 검증하기 위해 워킹셋 크기가 주기적으로 변화하는 워크로드를 구성하여 실험을 수행하였다. 실험에 사용된 워크로드는 총 4개의 구간으로 구성되며, 구간 1과 구간 3은 워킹셋 크기가 6G로 빠른 계층 메모리에 전부 수용할 수 있는 데이터에 50라운드씩 접근한다. 구간 2와 구간 4는 워킹셋 크기가 18G로 빠른 계층 메모리에 전부 수용할 수 없는 데이터에 20라운드씩 접근한다.

해당 시나리오에서 전체 메모리 대역폭을 비교 실험하였다. 비교 대상은 마이그레이션을 비활성화한 No migration과 TPP, TPP를 확장하여 구현한 제안 기법이며, 제안된 기법에서 α, ρ, λ, $P_{\max}$, $P_{\min}$은 각각 8, 0.4, 0.05, 3분, 1초로 설정하였다.

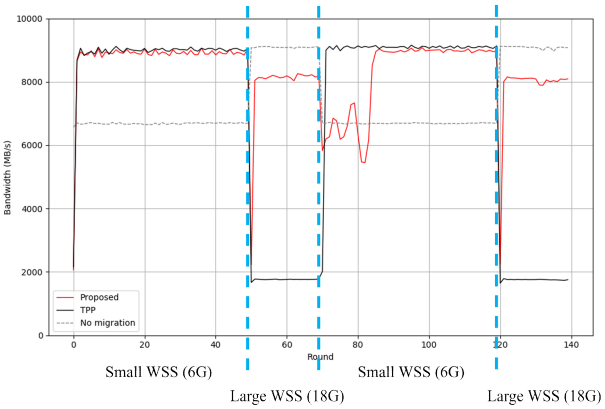


그림 6. 동적 워크로드 환경에서의 실험 결과

그림 6과 표 1, 표2는 제안 기법과 함께 TPP, No migration을 비교한 결과를 나타낸다. 작은 워킹셋 구간에서는 제안 기법과 TPP 모두 최대 9000MB/s, 9100MB/s가 넘는 높은 대역폭을 유지하나, No migration에서는 최대 6700MB/s의 비교적 낮은 성능을 보였다. 큰 워킹셋 구간에서는 더욱 명확한 성능 차이가 발생한다. 제안 기법과 No migration은 각각 최대 8000MB/s, 9000MB/s이 넘는 성능을 보이는 반면, TPP는 핑퐁 효과로 인한 불필요한 페이지 마이그레이션으로 최대 2000MB/s의 성능을 보였다.

한편 구간 3이 시작되는 지점(약 80라운드 전후)에서 제안 기법의 대역폭이 일시적으로 하락하는 현상이 관찰되었는데, 이는 이전 큰 워킹셋 구간에서 NUMA 스캔 주기를 늘림으로써 페이지 접근 불가 설정을 하지 않기 때문에 다시 승격되어야 하는 페이지를 식별하고 반영하는 데 일정 시간이 소요되었기 때문으로 해석된다.

그러나 제안 기법은 큰 워킹셋 구간에서 비효율적인 마이그레이션을 억제하기 위해 스캔 주

표 1. 각 기법의 구간별 평균 대역폭(MB/s)

	제안 기법	TPP	No migration
구간 1	8776.49	8849.58	6732.21
구간 2	8038.29	1778.21	8969.36
구간 3	8164.68	8941.55	6738.38
구간 4	8061.12	1757.71	9082.54

표 2. 각 기법의 구간별 최대 대역폭(MB/s)

	제안 기법	TPP	No migration
구간 1	9062.69	9116.02	6721,57
구간 2	8252.54	2015.74	9116.07
구간 3	9078.56	9155.13	6721,57
구간 4	8154.43	1790.13	9116.07

기를 자동으로 조절하고, 다시 작은 워킹셋 크기로 전환될 경우 빠르게 마이그레이션을 재개함으로써 효율적인 마이그레이션을 가능하게 한다. 그 결과, 동적 워크로드 환경에서 TPP 대비 최대 21.12%, No migration 대비 최대 13.22%의 성능 향상을 달성하였다. 이러한 결과는 워킹셋 크기의 변화에 적응하는 동적 NUMA 스캔 주기 조절 방식이 계층형 메모리 환경에서 페이지 마이그레이션 효율을 유의미하게 향상시킬 수 있음을 보여준다.

V. 결론

본 논문에서는 계층형 메모리 시스템에서 워킹셋 크기의 변화에 따라 페이지 마이그레이션을 동적으로 조절할 수 있는 기법을 제안하였다. 제안된 기법은 사용자 공간에서 측정된 로컬 메모리의 대역폭을 활용하여 워킹셋 크기의 변화를 감지하고, 이를 기반으로 커널 공간에서 NUMA 스캔 주기를 동적으로 조절한다. 워킹셋이 빠른 계층 메모리의 용량을 초과할 경우 페이지 마이그레이션을 억제하고, 워킹셋이 작아졌을 때는 마이그레이션을 활성화함으로써 핑퐁 효과와 대역폭 저하를 완화할 수 있도록 설계 및 구현되었다. CXL 기반 계층형 메모리 시스템을 모사한 환경에서의 실험을 통해 제안 기법의 성능을 평가한 결과, 대표적인 최신 기법인 TPP 대비 최대 21.12%의 성능 향상을 달성하였다. 이는 워킹셋의 특성을 반영한 마이그레이션 동적 주기 조절이 계층형 메모리 시스템의 효율성에 크게 기여할 수 있음을 보여준다.

REFERENCES

[1] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, “AI and memory wall,” *IEEE Micro*, vol. 44, no. 3, pp. 33-39, May 2024.

[2] Y. Zhong, D. S. Berger, C. Waldspurger, I. Agarwal, R. Agarwal, F. Hady, K. Kumar, M. D. Hill, M. Chowdhury, and A. Cidon, “Managing Memory Tiers with CXL in Virtualized Environments,” *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2024)*, no. 3, pp. 37-56, Jul. 2024.

[3] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan, “TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory,” *28th ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, vol. 3, pp. 742 - 755, Mar. 2023.

[4] Autonuma: the other approach to numa scheduling(2012). https://lwn.net/Articles/488709/.

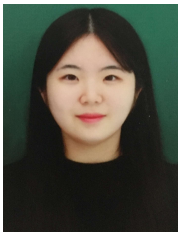
[5] T. Lee, S. K. Monga, C. Min, and Y. I. Eom, “MEMTIS: Efficient Memory Tiering with Dynamic Page Classification and Page Size Determination,” *29th ACM Symp. Operating Systems Principles (SOSP)*, pp. 17 - 34, Oct. 2023.

[6] A. Raybuck, T. Stamler, W. Zhang, M. Erez, and S. Peter, “HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM,” *28th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 392 - 407, New York, NY, USA: ACM, Oct. 2021.

[7] L. Xiang, Z. Lin, W. Deng, H. Lu, J. Rao, Y. Yuan, and R. Wang, “Nomad: Non-Exclusive Memory Tiering via Transactional Page Migration,” *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI'24)*, no. 2, pp. 19-35, Jul. 2024.

[8] 홍효림, 정형수, “계층형 메모리 시스템에서 페이지 마이그레이션 정책 최적화”, *서울대학교 공과대학 컴퓨터공학부 석사학위 논문*, 2025년 2월

저 자 소 개



이성민(정회원)

2024년 인하대학교 컴퓨터공학과 학사 졸업.
2024년~현재 인하대학교 전기컴퓨터공학과 석사과정 재학.

<주관심분야 : 지능형 임베디드 소프트웨어, 임베디드 시스템, 시스템 소프트웨어>

신지우(정회원)

2023년 인하대학교 컴퓨터공학과 학사 졸업.
2025년 인하대학교 전기컴퓨터공학과 석사 졸업.
2025년~현재 인하대학교 전기컴퓨터공학과 박사과정 재학

<주관심분야 : 지능형 임베디드 소프트웨어, AI 모빌리티>

정진만(종신회원)

2008년 서울대학교 컴퓨터공학과 학사 졸업.
2014년 서울대학교 전기컴퓨터공학과 박사 졸업.
2014년~2021년 한남대학교 정보통신공학과 부교수
2021년~현재 인하대학교 컴퓨터공학과 부교수

<주관심분야 : 운영체제, 임베디드 시스템, 시스템 소프트웨어>