NUMA 시스템의 공유 LLC 활용을 위한 I/O 트래픽에 따른 태스크 분류법

안득현, 김지홍, 엄영익
성균관대학교 정보통신대학
e-mail : {novum21, jjilong, yieom}@ece.skku.ac.kr

I/O Traffic based Task Classification for Shared Last Level Cache Utilization in NUMA Systems

Deukhyeon An, Jihong Kim, and Young Ik Eom
College of Information and Communication Eng., Sungkyunkwan University

요 약

디스크나 이더넷과 같은 I/O 장치로부터 발생하는 I/O 트래픽은, 여러 개의 노드를 가진 NUMA 시스템의 공유 LLC 에 캐시 오염을 일으켜 캐시 라인이 재사용되는 것을 방해한다. 이러한 태스크는 캐시를 효율적으로 이용할 수 있는 메모리 집중적인 태스크들과 따로 분리하여 다룰 필요가 있다. 본 논문에서는 이러한 캐시 오염을 발생시키는 태스크들을 해당 태스크의 I/O 트래픽을 이용하여 실시간으로 감시하고 분류하는 기법을 제안한다. 또한 대량의 I/O 트래픽을 일으키는 태스크의 특성을 알아본다. 이를 통해, NUMA 시스템 환경에서 각 노드의 공유 LLC 를 보다 효율적으로 사용할 수 있는 운영체제 스케줄링 기법을 연구하기 위한 토대를 마련하였다.

1. 서론

캐시 오염(Cache Pollution)은 기존의 캐시 라인(Cache Line)의 데이터가 새로운 데이터로 교체되어 캐시의 재사용이 방해되는 것을 말한다 [1]. 태스크(Task) 특성상 메모리나 I/O 집중적인 태스크는 캐시 오염에 큰 영향을 미친다. 정렬과 같은 일반적인 메모리 집중적인 태스크는 공간적, 시간적 지역성 특성을 가진 루프(Loop) 등에 따라 캐시를 재사용할 확률이 높다. 그러나 장시간 막대한 I/O 트래픽(Traffic)을 발생시키는 멀티미디어(Multimedia) 동영상, 웹상에서의 다운로드, 파일에 대한 처리와 같은 I/O 집중적인 태스크는 캐시의 재사용 확률이 낮아 심각한 캐시 오염을 일으킨다 [2]. 공유 LLC(Last Level Cache)를 가진 NUMA(Non-Unified Memory Architecture)시스템 환경의 노드(Node)에서 이러한 성향을 가진 I/O 집중적인 태스크와 메모리 집중적인 태스크를 함께 수행시킨다면, 캐시 오염으로 인해 메모리 집중적인 태스크의 캐시 효율성마저 저하되는 현상이 발생된다. 따라서 심각한 캐시 오염을 일으키는 I/O 집중적인 태스크를 잘 구분하여 다뤄야 할 필요성이 있다.

이 논문은 2012 년도 정부(교육과학기술부)의 재원으로 한국연구재단-차세대정보컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. 2011-0020520)

태스크가 캐시를 어떻게 이용하는지에 대한 특성을 파악하여, 이를 이용해 캐시 경쟁을 줄여 시스템의 성능을 향상시키고자 하는 연구들이 있다 [3][4][5]. 기존의 기법에서는 IPC(Instructions Per Cycle), 스택 거리차(Stack Distance), 캐시 미스(Cache Misses) 와 같은 방법들을 이용하여 태스크를 분류한다. 하지만 공유 LLC 를 가진 여러 개의 노드가 존재하는 NUMA 시스템 환경에서, I/O 트래픽에 따라 각 태스크가 공유 LLC 에 어떠한 영향을 미치는지를 고려하여 분류하는 연구는 없었다. 따라서, 본 논문에서는 심각한 캐시 오염을 일으키는 I/O 집중적인 태스크를 분류하는 기법을 다룬다.

본 논문은 특히 다음과 같은 부분에 이바지 하였다. 첫째, 메모리 집중적인 태스크와 I/O 집중적인 태스크가 서로 공유 LLC 에 어떠한 영향을 미치는지 확인하였다. 둘째, 막대한 I/O 트래픽을 일으켜 심각한 공유 LLC 오염을 일으키는 I/O 집중적인 태스크의 성향을 파악하고 이를 분류하는 기법을 제안하였다.

본 논문의 나머지 장은 다음과 같이 구성되어 있다. 2 장 관련연구에서는 기존의 연구를 간단히 소개하고 본 논문의 제안 기법과 비교하여 어떠한 차이점이 있는지 서술한다. 3 장에선 I/O 트래픽에 따른 태스크 분류법에 대하여 기술한다. 4 장에서는 본 논문의 자세한 실험환경에 대해 서술하고, 제안 기법을 마이크

로 벤치마크(Micro-benchmarks)를 통해 평가한다. 마지막으로, 5 장에서는 본 논문에 대한 결론과 향후 연구 계획에 대하여 토론하며 마친다.

2. 관련 연구

2.1 동물 분류법

Xie 와 Loh 가 제안한 동물 분류 기법은 각각의 태스크들이 캐시를 공유하는 코어쌍(Core Pair)에서 함께 수행되었을 경우, 서로에게 얼마나 영향을 미치는지를 파악하여 네 종류의 태스크로 나타낸 분류법이다 [3]. 각각 바다거북(Turtle), 양(Sheep), 토끼(Rabbit), 주머니곰(Tasmanian devil)으로 분류되며, 캐시 접근성과 캐시 미스율을 기준으로 한다.

본 논문에서는 동물 분류법과 달리, 분류과정에서 발생하는 부하를 줄이기 위해 두 종류로 태스크를 분류한다. 공유 LLC 크기 이상의 I/O 트래픽을 유발하여 캐시 오염을 발생시키는 I/O 집중적인 태스크와 그 외의 태스크로 나뉜다. 향후 캐시를 재사용하기 어려운 I/O 집중적인 태스크의 특성을 더욱 고려하여 이를 세밀하게 나누고자 한다.

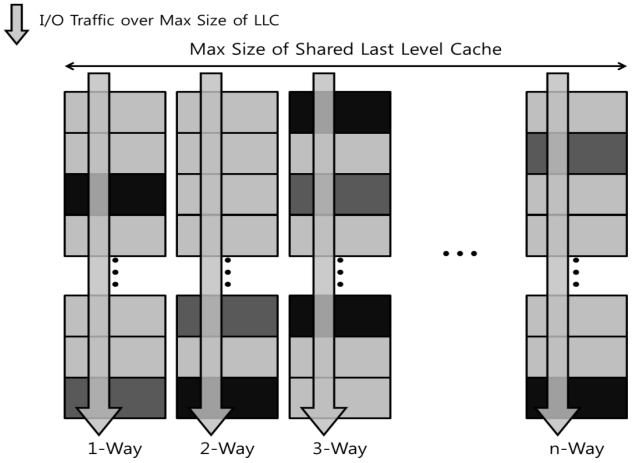
2.2 SDC 모델

Chandra 외 3 명은 여러 태스크들이 공유캐시에 미치는 영향을 예측하는 세 가지의 모델을 제안하였다 [5]. SDC(Stack Distance Competition)모델은 스택 거리차가 프로파일링되어 있는 태스크들을 함께 수행하였을 때, 각 태스크의 해당 프로파일(Profile)을 이용하여 새로운 스택 거리차 프로파일을 생성한다. 즉, 두 태스크의 공유 캐시에서 LRU(Least Recently Used) 스택 위치에 대해 경쟁한 상태를 보고 추가적인 캐시 미스를 추정한다. SDC 기법을 이용하여 태스크를 분류하면 캐시를 재사용하지 못하는 태스크를 보다 정밀하게 분류할 수 있지만, 그 과정이 복잡하여 본 논문에서 필요로 하는 것 이상의 추가적인 분류 비용(Cost)이 발생한다. 그러나 추후 스택 거리차를 이용하여 분류 비용이 효과적인 방법을 연구할 계획이다.

3. I/O 트래픽에 따른 태스크 분류법

공유 LLC 의 구조를 가진 멀티코어 프로세서에서 여러 개의 태스크를 동시에 수행시키면 캐시 오염이 발생한다 [1]. 특히 본 논문에서는 장시간 막대한 I/O 트래픽으로 인해 캐시 오염을 일으키며 캐시 재사용을 어렵게 만드는 I/O 집중적인 태스크를 구분하는 기법을 제안한다.

프로세서에 읽기 연산과 같은 I/O 명령이 요청되면 먼저 프로세서의 캐시를 확인한 뒤, 해당 정보가 없으면 메모리의 페이지 캐시(Page Cache)에서 검색하게 된다. 이 곳에서도 페이지 폴트(Page Fault)가 발생하면 비로소 HDD(Hard Disk Drive)나 SSD(Solid State Drive)같은 2 차 저장장치에서 해당 데이터 블록(Data Block)을 읽어오는데, 이 값 들은 다시 페이지 캐시와 프로세서의 캐시로 들어가게 된다. 이 때 페이지의 주소가 메모리에서 연속적으로 할당되면, 캐시 인덱스(Index)의 연속적인 값으로 인해 최악의 경우 거의 모든 재사용할 수 있는 캐시 라인이 대체(Replace)되어 될 수 있다.



(그림 1) LLC 크기에 따른 I/O 트래픽 측정

따라서, 본 논문에서는 이러한 캐시 오염을 최소화하기 위해 태스크의 I/O 트래픽을 실시간으로 감시하여, 그림 1 과 같이 해당 태스크의 I/O 트래픽이 공유 LLC 의 최대 크기를 넘어가면 *devil* 로 태스크를 분류하고, 그 이하로 트래픽이 떨어지면 *normal* 로 태스크를 분류한다. 이 때, 해당 태스크의 태스크 구조(Task_Struct)에 이를 구분할 수 있는 표시(Flag)를 설정한다.

캐시 오염을 일으킬 수 있는 I/O 장치에는 2 차 저장장치, 네트워크(Network) 등과 같은 다양한 것들이 있다. 현존하는 여러 운영체제들은 이러한 장치를 자신들의 방식으로 관리한다. 리눅스(Linux)에서는 모든 장치를 파일로 관리하며, 장치에서 I/O 트래픽이 발생할 때 커널(Kernel)은 해당 정보를 각 태스크 별로 루트(Root) 경로의 */proc* 경로 아래에 실시간으로 기록한다 [6]. 이렇게 커널 내부에서 제공하는 정보를 이용하여 I/O 트래픽을 실시간으로 측정하여 태스크를 분류하는데, 그 특성이나 작업 환경에 따라 트래픽이 갑작스럽게 변동되는 상황이 발생할 수도 있다. 이 경우, 태스크가 빈번하게 *devil* 이나 *normal* 로 분류되어, 불필요하고 추가적인 분류 부하(Overhead)가 발생할 수 있으므로 식 (1)과 같은 이동 평균(Moving Average) 공식을 사용하여 잦은 태스크 분류를 방지한다. 식 (1)에서 α의 값이 크면 클수록 현재의 I/O 트래픽이 태스크를 분류하는데 더욱 반영되고, 적으면 적을수록 이전의 I/O 트래픽을 값을 더욱 반영한다. 따라서, α값을 이용해 I/O 트래픽의 반영 정도를 조절할 수 있다.

$$(1) \quad MA = (\text{Previous I/O traffic} \times (1 - \alpha)) + (\text{Current I/O traffic} \times \alpha)$$

4. 평가 및 논의

이번 장에서는 본 논문에서 제안한 태스크 분류 기법을 평가해보고, 실험을 통해 캐시 오염을 유발시키는 *devil* 의 특성을 살펴본다. 실험 대상(Workload)

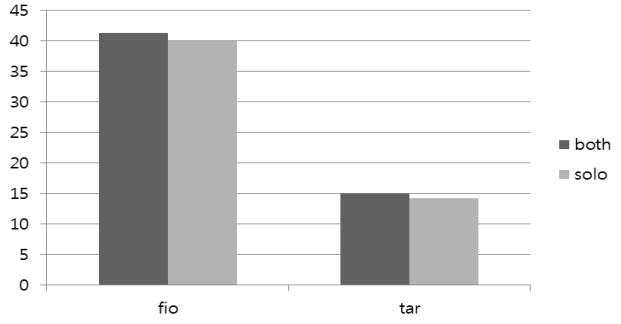
으로는 *devil* 의 성향을 가진 태스크와 메모리 집중적인 성향을 가진 태스크를 선정하였다. 실험은 NUMA 서버(Server)인 Dell PowerEdge T610 에서 리눅스 3.2.2 버전을 이용하였으며, 이에 대한 자세한 내용을 다음 표 1 에 나타내었다 [7].

<표 1> Dell PowerEdge T610 사양 및 실험 워크로드

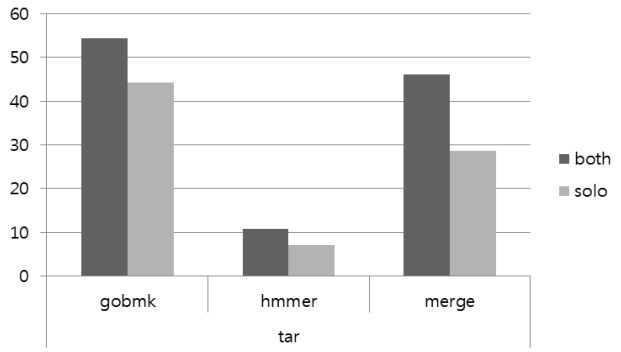
Component	Specification
L3 shared LLC	4MB, 16-way
Memory	2GB in each node
SSD	2 x Intel X25-E
I/O-intensive Task	fio, tar
Memory-intensive Task	gobmk, hmmer, merge(2MB)

Fio 와 **tar** 는 초당 수십 MB 이상의 I/O 의 트래픽이 발생되는 *devil* 워크로드이다. 워크로드 수행 시, 각각 다른 SSD 에서 수행되도록 하였는데, 이는 저장장치의 I/O 스케줄러에 의해 방해 받지 않고 동일하게 태스크를 수행할 수 있도록 하기 위함이다. 그림 2 를 보면 *devil* 태스크들끼리 같은 노드에서 함께 수행되어도 태스크의 공유 LLC 미스율에는 큰 영향이 없음을 알 수 있다.

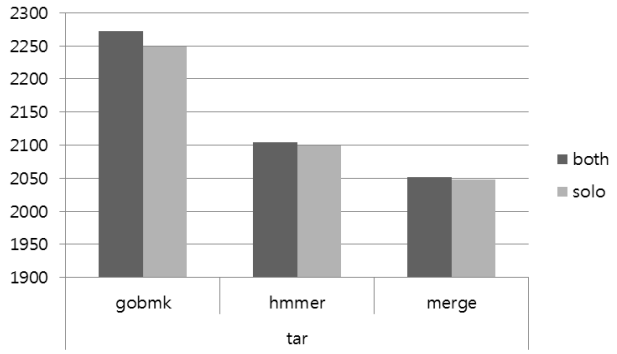
그림 3, 4 는 **tar** 와 메모리 집중적인 워크로드인 **gobmk**, **hmmer**, **merge** 를 같은 노드에서 동시에 수행하였을 때와 단독으로 수행하였을 때의 공유 LLC 미스율과 그에 따른 수행시간을 나타낸 그림이다.



(그림 2) fio 를 단독 및 함께 실행하였을 때 공유 LLC 미스율과



(그림 3) 메모리 집중적인 워크로드를 단독 및 tar 와 함께 실행하였을 때 공유 LLC 미스율



(그림 4) 메모리 집중적인 워크로드를 단독 및 tar 와 함께 실행하였을 때 수행시간

devil 에 의해 각 워크로드의 LLC 미스율, 수행시간이 늘어난 것을 알 수 있다.

5. 결론

본 논문에선 NUMA 시스템 환경에서 캐시 오염을 일으키는 I/O 집중적인 특성을 가진 태스크가 각 노드의 공유 LLC 에 어떠한 영향을 미치는지 알아보았다. 이러한 태스크를 I/O 트래픽에 의해 실시간으로 분류하여 공유 LLC 의 활용을 높일 수 있도록 하는 기법을 제안하였다. *Devil* 의 특성을 가진 태스크를 메모리 집중적인 태스크와 함께 수행하였을 때 추가적인 캐시미스율이 발생하였으며, 이에 따라 수행시간 또한 늘어나는 것을 확인하였다. 이러한 수행환경에서 본 논문의 제안 기법을 적용하여 태스크를 분류한다면, 캐시 효율성을 높일 수 있는 효과적인 스케줄링이 가능할 것으로 기대한다. 향후 본 제안기법을 보완하여 운영체제의 커널에 적용하려 한다.

참고문헌

[1] Soares, L., Tam, D., Stumm, M.: Reducing the Harmful Effects of Last-Level Cache Polluters with an OS-Level, Software-Only Pollute Buffer. In: IEEE MICROarchitecture, pp. 258-269. (2008)

[2] Ding, X., Wang, K., Zhang, X.: SRM-buffer: An OS buffer management technique to pre-vent last level cache from thrashing in multicores. In: 6th ACM European Conference on Computer Systems, pp. 243-256. (2011)

[3] Xie, Y., Loh, G.H.: Dynamic Classification of Program Memory Behaviors in CMPs. In: 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects, (2008)

[4] Zhuravlev, S., Blagodurov, S., Fedorova, A.: Addressing shared resource contention in multicore processors via scheduling. In: 15th ACM Architectural Support for Programming Languages and Operating Systems, pp. 129-142. (2010)

[5] Chandra, D., Guo, F., Kim, S., Solihin, Y., A.: Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture. In: 11th ACM High-Performance Computer Architecture, pp. 340-351. (2005)

[6] The Linux Kernel Archives: THE /proc FILESYSTEM, http://www.kernel.org/doc/Documentation/filesystems/proc.txt

[7] Jaleel, A.: Memory Characterization of Workloads Using Instrumentation-Driven Simulation, http://www.jaleels.org/ajaleel/workload/SPECanalysis.pdf