bperf, BCOZ를 활용한 응용의 On-CPU, Off-CPU 동시 성능 프로파일링

성균관대학교 ▌안민우
연세대학교 ▌정진규*

1. 서 론

최근 응용프로그램들의 실행 환경이 고도화됨과 동시에 복잡해짐에 따라 병목을 특정하고 최적화를 하기 위한 성능 프로파일링이 중요해지고 있다. 응용 프로그램의 성능 프로파일링은 개별 쓰레드의 성능 병목을 분석하고, 각 성능 병목의 개선이 응용 성능에 미치는 영향을 파악하는 것이 중요하다. 하지만, 현대의 멀티쓰레드로 구성된 응용프로그램의 성능 프로파일링은 두 가지 도전성을 내포한다.

첫 번째는, 기존 성능 프로파일링 기법들은 CPU에서 실행되는 작업(이하 on-CPU 작업) 정보에 기반하기 때문에 개별 쓰레드의 정확한 성능 병목을 파악하는 데 어려움이 있다는 것이다. 이는 개별 쓰레드의 실행이 비단 on-CPU 작업뿐 아니라 주변 장치(예, SSD, NIC, GPU/FPGA 등)의 실행 완료 대기, 쓰레드 간 동기화(예, lock)에 의한 대기, CPU 스케줄링에 의한 대기 등 다양한 CPU 외 작업(이하 off-CPU 작업)이 포함되어 있기 때문이다. 특히, 최신 응용프로그램들은 고성능/고용량 데이터 관리, 대규모 연산 제공 등 다양한 요구사항들이 존재하기 때문에 스토리지, 네트워크, 가속기와 같은 주변 장치 실행이 혼재하게 되어 off-CPU 작업 병목 분석의 중요성이 증가하고 있다.

두 번째는, 멀티쓰레드 응용프로그램의 경우 특정 쓰레드의 병목이 임계 경로(critical path) 상에 존재하지 않아 최적화가 무의미한 경우들이 존재하지만, 개별 병목이 응용프로그램 성능에 미치는 영향을 파악하기 어렵다는 것이다. COZ[1]는 가상 최적화(virtual speedup)을 통해 멀티쓰레드 응용프로그램의 개별 작업별 전체 성능에 미치는 영향을 분석해준다. 하지만, COZ의 가상 최적화 동작 또한 응용프로그램의 off-CPU 작업 정보가 없기에 on-CPU와 off-CPU 작업이 혼재하는 응용프로그램에 대한 정확한 성능 향상 예측이 불가능하다.

본 원고에서는, 기존의 성능 프로파일링 기법들이 간과하고 있는 off-CPU 작업 정보를 수집하는 샘플링 기반의 *blocked samples*[2] 기법을 소개한다. 또한, blocked samples 기반의 응용 성능 프로파일러인 *bperf* 및 *BCOZ*의 동작 핵심 원리 및 응용프로그램 프로파일링을 통한 실제 최적화 사례를 제공한다.

2. 성능 프로파일링 연구 동향

본 섹션에서는 off-CPU 작업 정보 수집 관점에서 리눅스 커널의 기존 성능 분석 도구 및 기존 프로파일러들을 비교 분석한다.

2.1 리눅스 성능 분석 도구

리눅스 운영체제는 그림 1[3]과 같이 내부 작업 정보를 관측할 수 있는 수 많은 도구를 제공한다. 응용(application)부터 운영체제 커널(operating system kernel)까지의 명령어(instruction) 실행 동작, 즉 on-CPU 작업은 리눅스 perf를 통해 분석이 가능하다. 이는 메모리 접근 명령어(load/store instruction)을 포함하기에 DRAM 접근 정보도 리눅스 perf의 관측 범위에 포함된다.

하지만, SSD와 NIC과 같은 주변장치로의 입출력 작업에 해당하는 off-CPU 작업은 리눅스 perf의 관측 범위에 포함되지 않는다. 리눅스에서 발생하는 입출력 작업은 iostat, netstat과 같은 system tap 도구를 통해 파악이 가능하다. 하지만, 해당 도구들은 리눅스가

* 종신회원
† 본 연구는 과학기술정보통신부의 재원으로 한국연구재단의 지원사업(RS-2023-00321688)의 연구 결과로 수행되었음.