클러스터 시스템에서 하드웨어 퍼포먼스 카운터 데이터 수집 방법 및 오버헤드 연구

박근철*, 박찬열*, 노승우*, 최지은*
*한국과학기술정보연구원
gcpark@kisti.re.kr, chan@kisti.re.kr, seungwoo0926@kisti.re.kr,
jieun1205@kisti.re.kr

Study on Hardware Performance Counter Data Collection Method and Overhead in Cluster System

Guenchul Park*, Chan-Yeol Park*, Seungwoo Rho*, Ji Eun Choi*
*Korea Institute of Science and Technology Information

요 약

대부분의 최신 마이크로 프로세서에서 사용 가능한 하드웨어 퍼포먼스 카운터는 시스템과 어플리케이션의 상태를 모니터링, 분석 및 최적화하는 다양한 용도로 폭넓게 사용되고 있다. 적은 오버헤드로 시스템의 가장 기본적인 정보를 수집할 수 있기 때문에 다양한 분야에서 활용이 가능하다. 이러한 퍼포먼스 카운터는 리눅스에 내장되어 있는 퍼프 이벤트를 통하여 수집 할 수 있는데 클러스터 시스템에서는 단일 노드에서와는 다른 방법을 사용하여 이벤트를 수집해야 한다. 본 연구에서는 클러스터 시스템에서 하드웨어 퍼포먼스 카운터를 수집하는 방법과 오버헤드에 대하여 연구하여 카운터의 활용을 지원하고자 한다.

1. 서론

현재 슈퍼컴퓨터는 페타스케일을 넘어 엑사스케일의 시대로 진입하는 단계에 있다. 가장 최근에 구축된 일본의 푸가쿠(Fugaku)는 0.5 엑사플롭스를 달성하였으며 슈퍼컴퓨터 강국인 미국과 중국은 엑사스케일 슈퍼컴퓨터 구축을 위하여 박차를 가하고 있다. 이러한 거대한 자본이 투입되는 엑사스케일 슈퍼컴퓨터를 효율적으로 활용하기 위해서는 시스템과 실행되는 어플리케이션을 효과적으로 분석하여 최적화하는 작업이 반드시 필요하다. 하드웨어 퍼포먼스 카운터(Hardware Performance Counter) 데이터는 이러한 분석의 중요한 기초 데이터로 활용된다.

하드웨어 퍼포먼스 카운터 데이터를 수집하는 방법은 크게 두 가지가 있다. 하나는 최신의 리눅스에 포함되어 있는 퍼프 이벤트(perf_event)를 사용하는 방법이다. 이 방법은 소스코드의 수정 없이 간단한 명령어로 어플리케이션 및 시스템의 하드웨어 퍼포먼스 카운터 데이터를 수집 할 수 있지만 멀티 노드로 구성된 클러스터 시스템에서는 수집이 어려운 단점이 있다. 다른 방법으로 Performance Application Programming Interface(PAPI)를 사용하여 수집하는 방법이 있다. 이 경우는 지원되는 프로그래밍 언어의 제약이 있으며 소스코드를 수정해야 하는 단점이 있다.

본 논문에서는 클러스터 시스템에서 퍼프 이벤트를 이용하여 하드웨어 퍼포먼스 카운터 데이터를 수집하는 방법을 제시하고 2개 노드로 구성된 클러스터 시스템에서 Nas Parallel Benchmark(NPB)의 하드웨어 퍼포먼스 카운터 데이터를 수집하였다. 그리고 실험을 통하여 클러스터 시스템에서 하드웨어 퍼포먼스 카운터 데이터 수집 시 발생하는 오버헤드를 측정 하였다.

2. 하드웨어 퍼포먼스 카운터

하드웨어 퍼포먼스 카운터는 최신 마이크로 프로세서에 내장되어 시스템과 프로세서의 성능을 모니터링하고 디버깅하기 위한 특별한 레지스터(x86에서는 Model-Specific Registers or MSR)의 집합이다. 마이크로 프로세서의 하드웨어 관련 동작 정보를 제공하므로 레지스터 및 카운터의 구성은 프로세서마다 상이 할 수 있으며 과거의 구형 프로세서에서는 제공되지 않는 경우도 있다. 최근의 거의 모든 프로세서는 카운터를 제공하고 있으며 사용자는 이러한 카운터를 저수준(Low Leve)의 성능 분석 및 최적화에 활용한다.

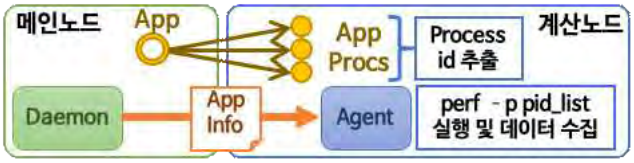
카운터를 수집하기 위해서는 서론에서 간략히 기술한 대로 두 가지 방법이 있다. 앞서 기술한 대로 PAPI는 여러 가지 제약이 있으며 시스템 및 프로세서에 따라 소스코드를 변경해야 하는 등의 문제가 있기 때문에 상세한 분석이 필요한 경우나 소스코드의 최적화에 활용된다. 퍼프 이벤트는 쉽게 사용 가능하고 바이너리 응용어플리케이션에 직접 적용이 가능하기 때문에 보다 많이 사용되고 있다. 리눅스 커널 2.6.31 이후 버전에는 퍼프 이벤트가 기본적으로 포함되어 있으며 perf 명령어를 통하여 쉽게 사용할 수 있다[1].

perf 명령은 기본적으로 카운터를 수집하고자 하는 응용어플리케이션의 앞에 덧씌워서 실행하는 형태로 사용한다. 이러한 방식은 단일 노드에서 실행되는 응용어플리케이션에 적용하기에는 문제가 없으나 MPI(Message Passing Interface)를 사용하여 다중 노드에서 어플리케이션을 실행하는 경우에는 문제가 될 수 있다. 다중 노드에서 카운터를 수집하기 위하여 아래와 같이 쉘 스크립트 파일을 사용하여 어플리케이션을 실행하면 카운터를 수집하지 않는 경우보다 작업시간이 길어지는 것을 볼 수 있다.

```
실행 명령 : mpirun  -n 64  -host h1,h2 job.sh
job.sh 파일 내용
  #!/bin/bash
  (perf stat  -e instructions /usr/bin/is.D.64)
```

이것은 MPI를 통하여 작업을 실행하는 mpirun에 perf와 응용어플리케이션이 순차적인 작업으로 전달되게 되고 이 경우 perf를 실행하는 프로세스들이 입출력 대기 상태로 기다린 후 응용어플리케이션이 실행되는 것으로 최종 수집되는 카운터 데이터에는 영향을 미치지 않는다. 하지만 이러한 지연으로 인하여 실시간으로 어플리케이션을 모니터링 하고 분석하기에는 적합하지 않다.

실시간으로 원격 계산노드에서 실행되는 어플리케이션을 모니터링하기 위해서는 perf의 '-p' 옵션을 사용하여 어플리케이션의 프로세스를 직접 모니터링 하여야 한다. 본 논문에서는 이를 위하여 파이썬을 사용하여 원격 계산노드의 모니터링을 수행하는 프로그램을 작성 하여 (그림 1)과 같은 방법으로 카운터 데이터를 수집하고 그 오버헤드를 비교 분석 하였다.



(그림 1) 원격 계산 노드 카운터 데이터 수집 방법

3. 원격 계산노드 카운터 데이터 수집 오버헤드 실험

본 장에서는 앞장에서 언급한 원격 계산노드에서 실행되는 응용어플리케이션의 카운터 데이터을 수집하는 프로그램을 사용하여 수집하는 경우 발생하는 오버헤드의 측정 실험을 진행하고자 한다.

3-1. 실험환경

실험에 사용된 시스템은 1개의 메인노드와 2개의 계산노드로 구성된 소규모 클러스터 시스템이다. 메인노드에서는 계산노드에 작업을 할당하는 역할만 수행하고 계산 작업에는 참여하지 않는다. 시스템의 사양은 <표 1>과 같다.

<표 1> 실험 시스템 사양

	메인노드	계산노드
프로세서	Intel(R) Xeon(R) CPU E5-2620 2,00GHz, 2 CPUs 12 cores	Intel(R) Xeon(R) Gold 6152 2.10GHz, 2 CPUs 44 cores
메모리	32GB(DDR3)	196GB(DDR4)
운영체제	CentOS 7.3	
커널	3.10.0-957.el7.x86_64	
파일시스템	Network File System	

본 논문의 실험에는 NASA에서 만든 전산 유체역학(CFD, Computational Fluid Dynamics)의 데이터 처리 및 계산에 관련한 8개의 응용어플리케이션으로 구성된 NAS Parallel Benchmarks(NPB)를 사용하였다[2]. NPB는 각각 다양한 특성을 가지는 어플리케이션이며 다양한 규모의 문제크기(S, W, A, B ,C, D, E, F)와 MPI, OpenMP 및 MPI+OpenMP 하이브리드를 통한 병렬 처리를 지원하여 많은 연구에서 벤치마크 어플리케이션으로 활용되고 있다.

본 실험에서는 NPB의 8개 응용어플리케이션을 실행시간을 고려하여 C 또는 D 클래스로 선정하여 실험하였다. 다양한 상황에서의 오버헤드를 측정하기 위하여 카운터 데이터를 수집하지 않는 경우와 수집하는 경우로 나누어 측정하였다. 카운터 데이터를 수집하는 경우는 2개의 항목을 수집하는 경우와 50개의 항목을 수집하는 경우로 나누어 실험하였으며, NPB 실행 전체의 카운터를 수집하는 것과 시간간격을 1초와 5초를 두고 반복적으로 카운터 데이터를 수집하는 실험을 수행하였다. 모든 실험은 100회를 실행하였고 결과는 그 평균값이다.

3-2. 실험 결과

실험의 결과는 <표 2>와 같다. 대부분의 NPB 어플리케이션에서 카운터 데이터를 수집하지 않을 때와 비교하여 2항목을 수집하는 경우 소폭의 실행시간 증가를 나타내었으며 평균 0.4%의 실행시간이 증가하였다. 50항목을 수집하는 경우에는 실행시간 증가 폭이 더 커져서 평균 1.5%의 실행시간 증가를 보였다. 인터벌을 통한 수집횟수의 차이는 거의 영향을 미치지 않는 것을 알 수 있다.

<표 2> 다양한 카운터 데이터 수집 방법에 따른 NPB 어플리케이션 실행 시간(초)

NPB	수집 없음	2 항목 수집			50 항목 수집		
		인터벌					
		없음	5초	1초	없음	5초	1초
BT-C	21.86	21.94	21.94	21.94	22.17	22.17	22.2
CG-C	6.33	6.39	6.41	6.39	6.48	6.47	6.45
EP-D	25.98	26.21	26.22	26.07	26.48	26.44	26.44
FT-C	6.77	6.81	6.8	6.81	6.82	6.8	6.82
IS-D	17.54	17.55	17.58	17.59	17.66	17.66	17.67
LU-C	15.55	15.62	15.6	15.62	15.77	15.76	15.8
MG-D	38.22	38.28	38.29	38.28	38.88	38.86	38.87
SP-C	20.53	20.61	20.6	20.61	20.87	20.89	20.87
합계	152.78	153.41	153.44	153.31	155.13	155.05	155.12
백분율	100	100.41	100.43	100.35	101.54	101.49	101.53
평균	100	100.4			101.52		

4. 관련연구

[3] 논문에서는 하드웨어 퍼포먼스 카운터 데이터 수집의 오버헤드에 대하여 연구하였으며 특히 PAPI와 같은 인터페이스를 사용하는 경우 보다 높은 오버헤드가 발생 할 수 있음을 밝혀내고 이를 해결하는 방안을 제시하였다.

[4] 논문에서는 하드웨어 퍼포먼스 카운터 기반의 퍼포먼스 분석 도구인 LIKWID의 오버헤드를 조사하고 이를 PAPI의 오버헤드와 비교하였다. 양쪽 모두 유사한 오버헤드 패턴을 보이고 데이터 집약적인 워크로드에서 보다 높은 오버헤드를 보이는 것을 확인하였다.

[5] 논문에서는 하드웨어 퍼포먼스 카운터 데이터를 이용하여 초고성능 컴퓨팅 시스템의 Uncore 에너지 효율을 최적화 하는 연구를 수행하였다. Uncore의 주파수를 조절하는 프로그램을 사용하여 1%의 성능 하락이 있지만 10%의 에너지를 절감 효과를 달성하였다.

5. 결론

본 연구에서는 하드웨어 퍼포먼스 카운터 데이터를 클러스터 시스템에서 효과적으로 수집하는 방법을 제시하고 카운터 데이터 수집의 오버헤드를 측정하는 실험을 진행하였다. 실험 결과 오버헤드는 수집 항목의 수에 영향을 받으며 인터벌과는 관련성이 없음을 보였다.

향후 다양한 초고성능 컴퓨팅 시스템용 마이크로프로세서를 대상으로 오버헤드 실험을 확대할 계획이며 보다 큰 규모의 클러스터에서 실험을 진행할 예정이다. 수집한 카운터 데이터를 기반으로 작업 실행 최적화 및 에너지 최적화의 연구도 수행할 계획이다.

Acknowledgement

이 논문은 2020년도 한국과학기술정보연구원(KISTI)의 주요사업 과제(No. K-20-L02-C08-S01) 및 정부의 재원으로 진행된 창의형 융합연구사업(No. G-19-GT -CU01-S01)의 지원을 받아 수행된 연구임.

참고문헌

[1] (Online) Perf Events Web-Page, http://web.eece.maine.edu/~vweaver/projects/perf_events/

[2] (Online) NPB, https://www.nas.nasa.gov/publications/npb.html

[3] V. M. Weaver, "Self-monitoring overhead of the Linux perf_ event performance counter interface," 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, 2015, pp. 102-111

[4] T. Röhl, J. et al. "Overhead Analysis of Performance Counter Measurements," 2014 43rd International Conference on Parallel Processing Workshops, Minneapolis, MN, 2014, pp. 176-185

[5] Neha Gholkar et al. “Uncore power scavenger: a runtime for uncore power conservation on HPC systems”, SC '19, Colorado, Denver, USA, 2019, pp. 1-23