KIPS Trans. Comp. and Comm. Sys.
Vol.9, No.10 pp.221~230
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2020.9.10.221

A Study on Scalability of Profiling Method Based on Hardware Performance Counter for Optimal Execution of Supercomputer

Jieun Choi† · Guenchul Park†† · Seungwoo Rho††† · Chan-Yeol Park††††

ABSTRACT

Supercomputer that shares limited resources to multiple users needs a way to optimize the execution of application. For this, it is useful for system administrators to get prior information and hint about the applications to be executed. In most high-performance computing system operations, system administrators strive to increase system productivity by receiving information about execution duration and resource requirements from users when executing tasks. They are also using profiling techniques that generates the necessary information using statistics such as system usage to increase system utilization. In a previous study, we have proposed a scheduling optimization technique by developing a hardware performance counter-based profiling technique that enables characterization of applications without further understanding of the source code. In this paper, we constructed a profiling testbed cluster to support optimal execution of the supercomputer and experimented with the scalability of the profiling method to analyze application characteristics in the built cluster environment. Also, we experimented that the profiling method can be utilized in actual scheduling optimization with scalability even if the application class is reduced or the number of nodes for profiling is minimized. Even though the number of nodes used for profiling was reduced to 1/4, the execution time of the application increased by 1.08% compared to profiling using all nodes, and the scheduling optimization performance improved by up to 37% compared to sequential execution. In addition, profiling by reducing the size of the problem resulted in a quarter of the cost of collecting profiling data and a performance improvement of up to 35%.

Keywords : Profiling, Scalability, Supercomputer, Hardware Performance Counter, Job Scheduling

슈퍼컴퓨터 최적 실행 지원을 위한 하드웨어 성능 카운터 기반 프로파일링 기법의 확장성 연구

최 지 은† · 박 근 철†† · 노 승 우††† · 박 찬 열††††

요 약

한정된 자원을 여러 사용자에게 공유해야하는 슈퍼컴퓨터와 같은 시스템은 응용프로그램의 실행을 최적화하는 방안이 필요하다. 이를 위해 시스템 관리자가 수행할 응용프로그램에 대한 사전 정보를 파악하는 것이 유용하다. 대부분의 고성능 컴퓨팅 시스템 운영에 있어 작업을 실행할 때 사용자로부터 실행 기간, 자원 요구사항들에 대한 정보를 제공 받거나 시스템 사용 통계 값을 사용하여 필요한 정보를 생성하는 등의 프로파일링 기술을 바탕으로 시스템 활용률을 높이는데 활용하고 있다. 본 논문의 선행연구에서는 하드웨어 성능 카운터를 이용하여 소스코드에 대한 별도의 이해 없이 응용프로그램 특성분석을 실행하고, 이 결과를 바탕으로 작업 스케줄링 알고리즘을 최적화하는 기술을 개발한 바 있다. 본 논문에서는 슈퍼컴퓨터 최적 실행지원을 위한 프로파일링 테스트베드 클러스터를 구축하고 구축한 클러스터 환경에서 하드웨어 성능 카운터를 기반으로 응용프로그램의 특성을 분석하는 프로파일링 기법의 확장성을 실험하였다. 이를 통해 응용프로그램의 문제크기를 축소하거나 프로파일링에 사용되는 노드수를 최소화하여도 개발한 하드웨어 성능 카운터 기반의 프로파일링 기법이 확장성 있게 동작하여 실제 스케줄링 최적화시에 활용될 수 있음을 보이고자 한다. 실험을 통해 프로파일링에 사용되는 노드의 수를 1/4로 줄여도 전체 노드를 사용한 프로파일링 대비 응용프로그램의 실행 시간이 1.08% 증가할 뿐 스케줄링 최적화 성능은 순차실행 대비 최대 37% 향상되었다. 또한 응용프로그램의 문제크기를 축소하여 프로파일링한 결과 프로파일링 데이터 수집 단계의 시간적 비용을 1/4배 이상 낮추면서 최대 35% 성능 향상 효과를 얻었다.

키워드 : 프로파일링, 확장성, 슈퍼컴퓨터, 하드웨어 성능 카운터, 작업 스케줄링

※ 본 연구는 2020년도 한국과학기술정보연구원 주요사업과 국가과학기술연구회 창의형 융합연구사업(No. CAP-17-03-KISTI, 차세대 초고성능컴퓨터를 위한 이기종 매니코어 하드웨어 시스템 개발)의 지원을 받아 수행된 연구임.
※ 이 논문은 2020년 한국정보처리학회 춘계학술발표대회의 우수논문으로 "응용 프로그램 특성 분석 기반 스케줄링 최적화 기법의 확장성 연구"의 제목으로 발표된 논문을 확장한 것임.
† 정 회 원 : 한국과학기술정보연구원 슈퍼컴퓨터기술개발센터 기술원
†† 정 회 원 : 한국과학기술정보연구원 슈퍼컴퓨터기술개발센터 책임연구원
††† 비 회 원 : 한국과학기술정보연구원 슈퍼컴퓨터기술개발센터 기술원
†††† 종신회원 : 한국과학기술정보연구원 슈퍼컴퓨터기술개발센터장
Manuscript Received : July 13, 2020
First Revision : August 20, 2020
Accepted : September 1, 2020
* Corresponding Author : Chan-Yeol Park(chan@kisti.re.kr)

1. 서 론

전세계 슈퍼컴퓨터의 계산 성능 상위 500개의 시스템을 공표하는 TOP500[1]에 따르면 2020년 6월 현재 상위 10위에 속하는 슈퍼컴퓨터들은 최소 280개에서 최대 158,976개의 계산 노드로 구성되어 있으며, 100 GB/s 이상의 전송속도를 갖는 고성능 인터커넥트 장비를 통해 대규모 클러스터 시스템 구조를 갖는다. 이와 같은 슈퍼컴퓨팅 시스템은 한정된 고성능 자원을 여러 사용자들에게 제공해야하기 때문에 제한된

기간 내에 보다 많은 양의 작업이 실행되도록 시스템 생산성을 높이는 방안이 필요하다. 이를 위해 시스템 관리자가 수행할 응용프로그램에 대한 사전 정보를 파악하는 것이 유용하다.

대부분의 고성능 컴퓨팅 시스템을 운영하는 기관들은 사용자로부터 작업 제출 시에 응용프로그램의 실행 시간, 자원 요구사항들에 대한 기본 정보를 제공받는다. 추가적으로 시스템 모니터링을 통해 자원 사용에 대한 모니터링 값이나 자원 상태 정보와 같은 통계 데이터를 수집하여 필요한 정보를 생성하는 등의 프로파일링 기술을 바탕으로 시스템 활용률을 높이기 위한 연구[2-13]를 진행하고 있다.

한편, 현대의 프로세서에는 하드웨어 관련 활동의 수를 기록하기 위한 특수 목적의 레지스터로 하드웨어 성능 카운터를 내장하고 있다. 여기서 하드웨어 관련 활동을 성능이벤트라 하며 성능이벤트는 하드웨어 수준의 정보로 활용되어 응용프로그램 및 시스템 성능분석을 위한 프로파일링 기술에 사용될 수 있다. 예를 들어 캐시 미스 이벤트는 CPU의 캐시 메모리 접근 요청이 실패한 활동을 말하며 하드웨어 성능 카운터에 캐시 미스가 발생한 횟수가 기록된다. 하드웨어 성능 카운터를 활용한 성능이벤트 수집은 응용프로그램의 소스코드 수정 없이 낮은 오버헤드로 응용프로그램이 실행하는 동안 컴퓨팅 시스템내의 하드웨어 동작 과정을 통찰 할 수 있다는 장점을 갖는다. 반면 프로세서 제조사별로 하드웨어 성능 카운터 수집을 위해 제공되는 레지스터의 개수와 성능이벤트의 명칭 등이 달라 프로세서 아키텍처에 대한 이해가 필요하다.

하드웨어 성능 카운터를 통한 성능이벤트 수집은 리눅스 perf[14]와 OProfile[15], PAPI[16] 등의 대표적인 데이터 수집 프로그램을 사용하여 수집할 수 있으며, 인텔 VTune [17]과 AMD uProf[18], IBM HPCS Toolkit[19] 등은 프로세서 제조사별로 제공되는 하드웨어 성능 카운터 기반 프로파일링 도구로 사용될 수 있다. 그러나 실제 사용자가 요청한 작업이 수행되어야할 대규모 고성능 컴퓨팅 시스템에서 작업의 프로파일링 데이터를 추출하기 위해 데이터 수집 프로그램을 실행하거나 성능 분석을 실행하는 과정은 시스템 활용률을 낮출 수 있으므로 실제 시스템과 유사하면서 적은 규모의 테스트베드 운용의 필요성이 대두된다.

본 논문의 선행연구[20]에서는 리눅스 perf를 사용하여 매니코어 형태의 컴퓨팅 시스템에서 응용프로그램을 실행하면서 성능이벤트의 일부를 기록하고 응용프로그램이 실행되는 동안 활용한 시스템 자원에 대한 특성을 도출하는 프로파일링 기법을 개발한 바 있다. 이를 바탕으로 본 논문에서는 슈퍼컴퓨터 최적 실행 지원을 위한 프로파일링 테스트베드를 구축하고 개발한 프로파일링 기법이 확장성 있게 동작함을 실험을 통해 보이고자 한다. 프로파일링 기법의 확장성 실험은 다음과 같이 3가지 방법으로 진행된다. (1)프로파일링 테스트베드 클러스터의 노드수를 다르게 사용하여 응용프로그램을 프로파일링하고, 그 프로파일링 데이터를 바탕으로 전체 노드를 사용하는 환경에서 스케줄링 최적화 성능 평가를 통해 제시한 프로파일링 기법의 확장성을 검증하였다. (2)응용프로그램의 문제크기를 줄여 프로파일링하고, 그 프로파일링 데이터를 바탕으로 대규모 응용프로그램으로 스케줄링 실험시 최적화 성능 평가를 통해 제시한 프로파일링 기법의 확장성을 검증하였다. (3)프로파일링 데이터 수집시 단일 노드 및 다중 노드에서 데이터를 수집하는 경우의 오버헤드를 측정하고 비용 대비 성능 향상의 효과를 분석한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 선행연구에서 개발한 하드웨어 성능 카운터 기반의 프로파일링 기법과 자원 간섭률 기반 스케줄링 알고리즘에 대해 간략하게 소개하고 관련 연구들을 살펴본다. 3장에서는 구축한 프로파일링 테스트베드를 소개하고 4장에서는 프로파일링 기법의 확장성 실험 내용을 다룬다. 마지막으로 5장에서 향후 연구를 제시하며 본 연구의 결론을 맺는다.

2. 관련 연구

2.1 하드웨어 성능 카운터 기반 프로파일링 기법

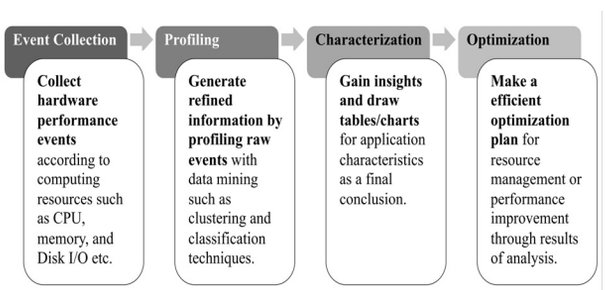


Fig. 1. Profiling Method Based on Hardware Performance Counter

하드웨어 성능 카운터 기반 프로파일링 기법은 Fig. 1과 같이 이벤트 수집, 프로파일링, 응용프로그램 특성 분석 및 최적화의 4단계로 이루어져있다. 먼저, 하드웨어 성능 카운터를 수집하는 이벤트 수집 단계에서 성능을 분석할 시스템의 프로세서를 기반으로 하드웨어 성능 카운터 및 이벤트를 식별하고 자원 범주(CPU, Memory, I/O)에 따라 성능이벤트를 분류하는 과정을 수행한다. 이후 시스템에서 응용프로그램을 실행하면서 선별된 이벤트에 대한 데이터를 수집하고, 수집된 원시데이터는 프로파일링 단계에서 데이터마이닝 과정을 거쳐 정제된 정보로 재생성된다. 프로파일링 단계에서 도출된 정보는 응용프로그램 특성화 단계에서 표나 차트로 시각화되고 더 나아가 시스템 최적화 전략의 발판으로 사용될 수 있다.

선행 연구[20]에서는 인텔 제온파이 프로세서를 탑재한 단일 노드에서 응용프로그램을 실행하면서 성능이벤트를 수집하고, 기댓값-최대화 군집분석을 통해 응용프로그램별 자원 범주에 따른 상대적인 자원 사용량에 대한 성능프로파일을 생성하였다. 도출된 정보는 Fig. 2와 같이 키비아트 차트를 활용하여 도식화된다. 차트에서 삼각형의 각 꼭짓점은 CPU(C), Memory(M), I/O(I) 자원을 나타내고, 삼각형의 크기는 응용프로그램별 상대적인 자원 사용률에 따라 자원 사

용이 낮은 군집(1), 중간 군집(2), 높은 군집(3)에 속함을 보여준다. 예를 들어 Fig. 2에서 응용프로그램1의 경우 계산 자원의 사용이 응용프로그램2에 비해 상대적으로 높게 나타났으며 메모리, I/O 자원의 사용은 가장 낮은 군집에 속하는 것으로 특성을 보여준다.

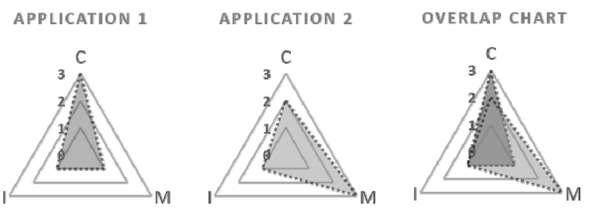


Fig. 2. Kiviat Charts for Characteristics Analysis of Application

두 개의 응용프로그램 특성화 차트를 중첩시키면 Fig. 2에서 가장 오른쪽의 중첩 차트에서 볼 수 있는 바와 같이 중첩된 면적이 발생하고, 이를 응용프로그램간의 자원 간섭률로 계산가능하다. 최적화 단계에서는 계산된 자원 간섭률을 고려한 작업 스케줄링 기법을 제시하여 전체 작업 시간을 줄이는 시스템 최적화 기술을 보인 바 있다. 본 논문에서는 다중 노드의 클러스터로 실험 환경을 확장하여 응용프로그램의 문제크기와 프로파일링에 사용되는 노드 수를 변경하며 프로파일링하고, 그 프로파일링 결과의 효용성을 검증함으로써 프로파일링 기술의 확장성을 보이고자 한다.

2.2 고성능 컴퓨팅 시스템 프로파일링 연구 동향

대부분의 슈퍼컴퓨터를 운영하는 기관들은 고성능 컴퓨팅 시스템의 활용률을 높이기 위해 다양한 목적을 가지고 프로파일링 기술 연구를 진행하고 있다.

대표적으로 프로파일링 데이터 수집 과정에서의 오버헤드를 줄이기 위한 연구[2-5]가 진행되고 있다. 중국에서는 TaihuLight 슈퍼컴퓨터에서 실행되는 응용프로그램의 I/O 동작을 프로파일링하기 위해 1% 이하의 낮은 오버헤드를 갖는 Beacon[2] 시스템을 개발하였다. Beacon은 I/O 전송노드나 버스트버퍼와 같은 추가적인 I/O 계층을 포함하는 슈퍼컴퓨터 시스템에서 효과적인 I/O 데이터 수집 도구로 사용가능하다[2, 3]. IBM BG/Q Mira 슈퍼컴퓨터와 그 테스트베드 시스템인 Cetus에서 응용프로그램의 MPI 특성을 분석한 연구[4]에서는 경량의 프로파일링 도구인 Autoperf의 오버헤드를 분석하였다. Autoperf의 경우 지연시간에 민감한 적은 규모의 메시지를 주고받는 MPI 연산에 대해서 0.2 μs 정도의 오버헤드를 보였으며, 512 bytes 이상의 메시지 크기에 대해서는 성능 하락이 거의 발생하지 않는다. 이처럼 프로파일링 데이터 수집 과정에서의 오버헤드를 줄이기 위한 연구뿐만 아니라 공용 클라우드를 활용한 프로파일링 데이터 수집의 이점을 제시하는 연구[5]도 진행되었다. 이 논문에서는 실제 서비스되는 고성능 컴퓨팅 시스템의 성능에 영향을 전혀 미치지 않고, 응용프로그램의 다양한 I/O 요구사항과 I/O 노드 수 조절 및 I/O 관련 파라미터 설정을 다양하게 실험할 수 있다는 점에서 공용 클라우드를 활용한 가상 클러스터 테스트베드 운영의 장점을 말한다.

또한 방대한 양의 데이터를 처리할 수 있는 확장성 있는 분석 프레임워크가 필요함을 강조하며 기존의 고성능 컴퓨팅 시스템 모니터링 도구와 빅데이터 분석 플랫폼을 결합한 프레임워크가 제시하는 연구가 진행되었다[6]. 이 프레임워크는 Titan 슈퍼컴퓨터에 적용되도록 설계되어 컨테이너 기술 기반으로 로그의 양이나 시스템 상황에 따라 스케일 인-아웃 방식으로 확장성 있게 동작한다.

최근에는 계산 노드의 프로파일링 기술뿐만 아니라 I/O[7], 네트워크[8] 및 전력[9, 10] 중심의 프로파일링 기술 분석 및 최적화 연구가 활발하게 진행되고 있다. 오크릿지 국립 연구소에서는 Summit 슈퍼컴퓨터의 두개의 I/O 계층(In-system storage layer, Spider3 parallel file system layer)을 고려한 I/O 서브시스템(Spectral, SymphonyFS,)의 성능 평가를 위해 프로파일링 데이터를 수집하고 서브시스템별 적합한 I/O 패턴과 기계학습 워크로드의 처리 성능 등을 분석하여 최적의 사용법을 제시하고 있다[7]. 중국과 미국 사이의 데이터 배치 연구를 위해 대륙간 테스트베드 시스템을 제시한 논문[8]에서는 네트워크 프로파일링 기술을 통해 데이터 전송 방법에 따른 네트워크 성능 특성화를 진행하였다. 시스템의 전력 및 에너지 프로파일링 관련 연구들은 데이터를 기반으로 응용프로그램의 전력이나 실행 시간 등을 예측하여 주파수를 조절[9]하거나 주어진 전력 제한 내에서 작업 스케줄링이 가능한 알고리즘[10]을 개발하고 있다.

한편, 프로파일링 도구들이 실제 서비스 되고 있는 슈퍼컴퓨터에서 동작함으로써 발생하는 오버헤드를 줄이고자 보다 적은 규모의 프로파일링 테스트베드를 운용하고 여기서 얻은 프로파일링 데이터를 기반으로 스케줄링 알고리즘을 개선하는 연구가 진행되고 있다. 고성능 컴퓨팅 시스템에서 작업에 대한 정보 및 통계 값을 사용하여 생성된 대략적인 프로파일 데이터를 활용하는 연구[11]에서는 오프라인 스케줄링을 통해 자원 활용률 기반으로 프로파일을 재생성한 다음 수정된 프로파일을 기반으로 스케줄링 기법을 제안하였다. 제안된 스케줄링 기법은 CINECA센터에서 운영하는 EURORA 슈퍼컴퓨터의 작업 스케줄러인 PBS[21] 스케줄러 성능을 개선시켰다. 동적 자원 관리 플랫폼을 제안한 연구[12]에서는 스케줄러가 사전에 노드수와 전력 사용 레벨의 조합에 대한 성능 프로파일 데이터 셋을 저장하도록 한다. 즉, 스케줄러는 프로파일이 없는 새로운 작업에 대해서 필수 전력특성을 바탕으로 성능 모델링을 통해 자원을 할당한다. 제안된 기법은 아르곤 국립연구소의 IBM BG/P 시스템에서 선입선출 및 backfiling 스케줄링 기법을 사용하는 Slurm[22] 스케줄러와 비교하여 성능을 최대 5.2배 향상시켰다. 전력 프로파일링 생성 방법을 연구한 논문[13]에서는 작업 모니터링 데이터 수집, Cray 플랫폼을 활용한 out-of-band 데이터 수집,

하드웨어 성능 카운터를 활용한 In-band 데이터 수집 및 코드 인스트루멘테이션 기반의 코드 프로파일링 데이터 수집 방법을 비교 분석한다. 이 연구는 소규모 테스트베드에서 생성된 전력 프로파일을 이용하여 로스앨러모스 국립연구소의 Trinity 슈퍼컴퓨터를 대상으로 프로파일링 데이터의 확장성 실험을 진행하였다.

3. 슈퍼컴퓨터 최적 실행 지원을 위한 프로파일링 테스트베드 구축

국가 슈퍼컴퓨터 5호기 누리온[23]은 지난 2018년에 도입되어 1년 여간 140개 기관과 2000여명이 넘는 연구자들을 대상으로 서비스되었다. 누리온은 인텔 제온파이 프로세서 기반의 계산노드 8,305대와 인텔 제온 프로세서 기반의 CPU-only 노드 132대로 구성되어 총 25.7 페타플롭스 이론 성능을 갖추었다. 누리온은 작업 스케줄러로 PBS[21]을 사용하고 있으며, 상용 소프트웨어를 사용하지 않는 일반 작업에 대해서는 배타적 노드 할당 정책을 적용하여 노드 당 하나의 작업만이 배치된다. 이렇게 운영되고 있는 누리온의 생산성을 높이기 위해서 한국과학기술정보연구원 국가슈퍼컴퓨팅본부에서는 사용자가 실행할 응용프로그램을 슈퍼컴퓨터 규모에 맞춰 최적 실행될 수 있도록 소스코드의 최적화 및 병렬화 기술을 지원하고 있다[23]. 본 연구에서는 누리온의 최적 실행 지원을 위해 응용프로그램의 성능 프로파일링이 가능한 테스트베드 클러스터를 구축하였다.

Table 1. Specifications of KISTI-GVP Node

Platform	KISTI-GVP(Groveport)
Processor	Intel Xeon Phi CPU 7290@1.50GHz, 72 Cores (288 Cores by hyper-threading)
Memory	192 GB (DDR4), 16 GB (MCDRAM)
Network	Melanox Infiniband 100GB/s
OS	CentOS 7.3
Kernel	3.10.0-514.el7.x86_64
Cluster Mode	Quadrant Mode
Memory Mode	Cache Mode
File system	Network File System

프로파일링 테스트베드 클러스터는 누리온의 계산노드와 동일한 인텔 제온파이 프로세서 기반의 'KISTI-GVP'서버 16노드로 구축하였다. KISTI-GVP서버는 한국과학기술정보연구원 슈퍼컴퓨터기술개발센터에서 진행 중인 창의형 융합연구 사업과제의 연구 결과인 자체개발 메인보드에 인텔 제온파이 프로세서를 장착한 시제품으로, Table 1은 KISTI-GVP서버의 하드웨어 및 소프트웨어 세부 사양을 보여준다. 인텔 제온파이 프로세서 기반 시스템은 온-다이 통신을 위해 칩을 가상의 영역으로 분리하는 방법에 따라 all-to-all, quadrant, SNC의 세 가지 클러스터 모드를 제공한다. 또한 온-패키지 메모리인 MCDRAM(Multi-Channel DRAM)을 고대역폭 메모리 또는 캐시로 활용함에 따라 메모리 모드를 flat, cache, hybrid로 설정 가능하다. 국가 슈퍼컴퓨터 5호기 누리온은 일반적으로 quadrant 클러스터 모드와 cache 메모리모드를 사용하기에 프로파일링 테스트베드 역시 동일하게 quadrant와 cache 모드로 구성하여 실험을 진행하였다.

4. 하드웨어 성능 카운터 기반 프로파일링 기법의 확장성 연구

본 장에서는 KISTI-GVP 테스트베드 클러스터에서 하드웨어 성능 카운터 기반으로 응용프로그램의 성능을 프로파일링하고, 응용프로그램의 문제 크기와 프로파일링에 사용되는 노드수에 따라 프로파일링 기법의 데이터 생성에 소요되는 시간을 분석한다. 또한 프로파일링 기반 작업 스케줄링 최적화 실험을 통해 프로파일링 기법이 확장성있게 동작함을 보인다. 한편, 실험을 진행하기에 앞서 실험에 사용된 응용프로그램에 대해 설명하고자 한다.

4.1 NPB 응용프로그램

실험에 사용된 응용프로그램은 NASA에서 개발한 전산유체역학분야의 계산 및 데이터 연산을 벤치마크한 Nas Parallel Benchmark(NPB)[24] 프로그램을 대상으로 하였다. NPB는 5개의 커널 프로그램(IS, EP, CG, MG, FT)과 3개의 수도 어플리케이션(BT, SP, LU)으로 구성되어 있다. 또한 I/O 연산 벤치마크를 위한 프로그램(BT_epio, BT_full) 등을 포함한다. 각 프로그램은 가장 작은 규모의 문제크기인 S, W 클래스부터 A, B, C, D, E, F 클래스로 규모를 변경하며 실행이 가능하다. 한편, NPB 프로그램은 MPI를 사용해서 병렬로 실행할 수 있다. 이때 실행 프로세스의 수는 2의 승수 또는 n의 2승에 해당하는 수로 제한되어 있다. 본 실험에서는 노드 당 MPI 프로세스 수(PPN)는 하나의 물리서버가 갖는 최대 물리 코어 수(72)를 고려하여 설정하였다.

Table 2는 10개의 NPB 응용프로그램을 대상으로 PPN을 64로 설정했을 때, 단일노드 및 다중노드에서의 문제크기별 실행 시간 측정 결과를 보여주고 있다. 이어지는 프로파일링 확장성 실험에서도 단일노드 프로파일링의 경우 위의 표와 같이 문제크기를 C 또는 D클래스를 사용한다. 이는 문제크기가 너무 클 경우 단일노드에서 실행이 불가능한 경우가 있고, 응용프로그램 특성 분석에 있어 실행 시간이 미치는 영향을 줄이기 위함이다. 마찬가지로 다중노드 프로파일링의 경우 노드의 수를 4, 8, 16으로 설정하고, 해당 규모의 클러스터에서 실행하기 적합한 D 또는 E클래스의 프로그램을 대상으로 하였다. 본 논문에서는 다중노드 프로파일링에서 사용한 응용프로그램 대비 단일노드에서 실행한 응용 프로그램의 규모가 작다는 의미로 단일노드에서 실행한 프로그램들은

Table 2. Execution Time According to Problem Size of NPB Programs in Single Node and Multiple Nodes

NPB	Single		Multi			
	class	time	class	4 nodes	8 nodes	16 nodes
BT	C	50.29	D	260.20	163.78	133.00
EP	D	32.60	E	124.78	64.74	34.12
LU	C	29.93	D	135.72	78.19	49.12
SP	C	34.13	D	196.67	144.32	125.61
IS	D	31.88	E	207.46	147.04	189.45
MG	D	68.22	E	213.94	84.62	37.51
FT	C	10.26	D	134.45	116.26	155.57
CG	C	9.54	D	148.61	79.41	53.16
BT_epio	C	75.02	D	385.79	285.99	207.76
BT_full	C	118.94	D	891.82	712.10	605.58
Total (seconds)		460.81		2699.49	1876.48	1590.93

'소규모 응용프로그램'이라 하고, 다중노드에서 실행한 프로그램들은 '중규모 응용프로그램'이라 표현한다.

4.2 응용프로그램 문제 크기 및 프로파일링 노드 수에 따른 프로파일링 성능 분석

슈퍼컴퓨터 최적 실행 지원을 위한 프로파일링 테스트베드 운영 시에 실제 사용자가 실행할 수백 노드의 클러스터 규모의 작업을 직접 프로파일링하지 않는다. 즉, 실제 실행환경과는 다른 적은 규모의 응용프로그램을 몇 개의 노드를 사용해서 프로파일링할 것인지 결정하는 것이 중요하다. 본 실험에서는 2.1절에서 소개한 하드웨어 성능 카운터 기반 프로파일링 기법에 따라 NPB 응용프로그램을 대상으로 단일노드에서 소규모 응용프로그램을 프로파일링하고, 다중노드에서 중규모 응용프로그램을 프로파일링하여 그 성능을 분석하였다.

1) 성능이벤트 수집 단계

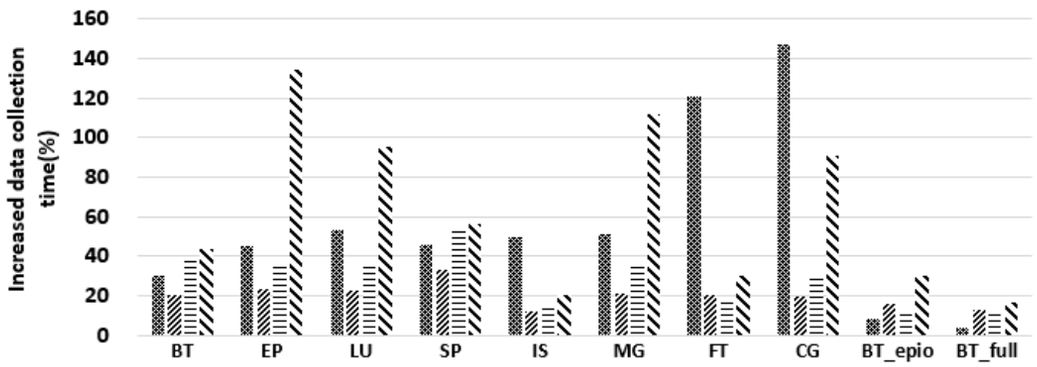
이벤트 수집 단계에서 각 응용프로그램별로 리눅스 perf 명령어를 사용하여 실행 시간과 40개의 성능이벤트의 발생횟수를 기록한다. Table 3은 단일 노드 프로파일링 시에 이벤트 수집단계에서 생성된 원시 데이터의 일부를 나타낸다. 수집된 데이터는 실행시간을 기준으로 정규화 되어 프로파일링 단계에서 입력 값으로 사용된다.

한편, 성능이벤트 수집 시 단일노드의 경우 작업이 끝나면 리눅스 perf를 통해 출력되는 결과 값을 기록하면 되지만 다중 노드에서는 자식 프로세스로 생성되는 MPI 프로세스 전부에 해당하는 성능이벤트 값을 수집하기 위해서 MPI를 실행하는 mpirun 프로세스가 아닌 각각의 서브 프로세스의 성능이벤트 값을 수집하는 것이 필요하다. 여기서 단일 노드에서 사용하는 방식을 'N-master'라 하고 자식 프로세스 각각

Table 3. Sample Data from the Event Collection Step Generated by Single Node Profiling

Raw data					
No.	elapsed_time(s)	instructions	branch-instructions	branch-misses	iTLB-loads
bt.c	51.383186197	3857719245835	93619036996	1252742735	3857435020399
ep.d	33.795426178	2119918534547	150292718446	3383969353	2119388488253
lu.c	31.099412799	1789620962853	98874655227	1205293843	1789205758789
sp.c	35.455802488	1745064712225	94311159313	3047210748	1744855917385
mg.d	84.235786772	1294573479621	75238369384	2205757189	1293889185429
is.d	34.921278906	1008160692700	95835607962	249644603	1007933928061
ft.c	10.392682052	397059125336	53111479682	575260227	396732903487
cg.c	10.938589096	342770945836	44096590959	2956542271	342461255686
bt_epio	76.440949583	4645930536233	268242334772	2136262120	4646899441242
bt_full	120.232949138	8931277304597	1427935564927	5417978390	8931027210546

Normalized data					
No.	elapsed_time(s)	instructions	branch-instructions	branch-misses	iTLB-loads
bt.c	100	7507746271366	182197804233	2438040198	7507193122648
ep.d	100	6272797163092	444713191822	10013098621	6271228766556
lu.c	100	5754516892091	317930939296	3875616081	5753181805575
sp.c	100	4921802891925	265996403111	8594392270	4921214004281
mg.d	100	1536845002859	89318770877	2618551180	1536032647187
is.d	100	2886952380575	274433270959	714878179	2886303021083
ft.c	100	3820564541106	511046902217	5535243200	3817425583835
cg.c	100	3133593764474	403128690286	27028552269	3130762593586
bt_epio	100	6077803273736	350914446033	2794656701	6079070794631
bt_full	100	7428310931952	1187640804929	4506234297	7428102924037



	BT	EP	LU	SP	IS	MG	FT	CG	BT_epio	BT_full
1N-process	30.01	45.25	53.09	45.85	49.75	51.51	120.76	147.27	8.61	3.98
4N-process	20.76	23.79	23.02	33.14	12.13	20.90	20.75	19.79	16.13	12.84
8N-process	38.28	37.07	34.42	54.37	13.52	34.75	17.91	29.20	12.98	10.63
16N-process	43.70	134.22	95.13	56.70	20.74	111.59	29.87	90.92	30.55	16.59

Fig. 3. Increase Data Collection Time in '#N-process' Compared to 'N-master'

의 성능이벤트 값을 수집하는 방식은 '#N-process'라 한다. N앞에 붙는 숫자는 노드의 개수를 의미한다.

#N-process 방식의 다중노드 프로파일링 시에는 노드별로 수행한 프로세스의 perf 데이터를 수집하기 위해 프로세스 별로 perf 데이터를 파일로 기록하도록 쉘 프로그램을 이용하고, 쉘 프로그램을 통하여 각각의 프로세스에서 수집된 perf 데이터를 파일로 저장하도록 하였다. 즉, 16노드에서 PPN 64로 MPI 프로그램을 실행하는 경우 MPI 작업 프로세스는 1024개 생성되고 perf 결과도 1024개의 파일로 생성된다. 응용프로그램의 실행이 종료되면 1024개의 파일은 취합되어 응용프로그램별 다중노드에서 발생한 성능 프로파일링 값이 Table 3의 단일 노드에서의 결과와 동일하게 하나의 파일로 생성된다. 그러나 성능이벤트 값을 프로세스별로 추출하는 것은 프로파일링 작업에 있어 데이터 측정에 걸리는 실행 시간의 차이를 불러일으킬 수 있다.

Fig. 3은 프로파일링에 사용된 노드 수에 따라 N-master 대비 #N-process 방식의 증가된 실행 시간을 나타낸다. 응용프로그램별로 막대그래프는 단일노드, 4노드, 8노드, 16노드를 사용했을 때 프로파일링 원시데이터 생성에 소요된 시간을 perf elaspsed time을 측정해서 계산하였다. N-master의 경우 이는 Table 2에 측정된 값과 같다. '#N-process'는 다중노드 프로파일링에서 사용된 측정 방법에 따라 각 프로세스가 perf 로그를 남기고 가장 실행 시간이 길게 측정된 프로세스 로그에서의 perf elapsed time의 값이다.

Fig. 3에 따르면 프로세스별 기록 없는 N-master 측정 에 비해 프로세스별 perf데이터를 가져오는 #N-process 방법은 전체 평균값을 볼 때 1노드 55%, 4노드 20%, 8노드 28%, 16노드 63%로 실행 시간이 높게 나타난다. 이러한 실행 시간의 차이는 perf를 여러 노드에 걸쳐 병렬로 실행하기 위해서 MPI 내에서 쉘 프로그램으로 perf 명령어를 사용하기 때문이다. 즉, #N-process의 방식을 살펴보면 가장 먼저 각 프로세스에 매칭 되는 perf 프로세스들이 I/O 대기 상태(D)로 실행되고 이후 NPB 응용프로그램이 실행되는데 이때 perf 프로세스들의 대기시간에 따른 perf elapsed time 차이가 발생하였다. 그러나 앞에서 언급했다시피 단일노드의 경우 1N-process 방법을 사용할 필요 없이 1N-master 값을 취하면 되기에 단일노드 프로파일링 데이터 수집 방법은 다중노드 프로파일링 데이터 수집에 비해 실행시간의 차이가 없다고 볼 수 있다.

Fig. 3에서 NPB 10개의 응용프로그램들의 1N-master 값의 전체 실행 시간은 460.81초로 16N-process의 2152.75초에 비해 78% 적게 나타난다. 이는 단일 노드에 비해 다중 노드 프로파일링 데이터 수집에 따른 실행 시간 차이가 발생할 수 있음을 보여준다. 또한 다중노드 프로파일링 데이터 수집의 경우 노드수가 증가할수록 실행시간의 증가폭이 크다. 16노드의 경우 4노드에 비해 원시 데이터 생성에 드는 시간적 비용이 3배 증가하는 것을 알 수 있다.

2) 프로파일링 단계

Table 4는 단일 노드 및 다중 노드에서 프로파일링 결과 생성된 데이터를 보여준다. 이를 Fig. 2의 키비아트 차트와 함께 보면 Table 3의 C, M, I 지표는 차트의 꼭지점에 해당하는 각 CPU, Memory, I/O 자원 범주를 의미한다. 또한 표에 표시된 1, 2, 3의 숫자는 응용프로그램별 자원 범주에 따라 상대적으로 자원 사용이 높은 군집(3), 중간 군집(2), 낮은 군집(1)으로 분석된 것을 나타낸다. 예를 들어 8노드를 사용하여 중규모 NPB 프로그램을 대상으로 프로파일링한 결과 10개의 NPB 프로그램은 {BT, EP, LU(C:2, M:1, I:1)}, {SP(C:3, M:1, I:1)}, {IS(C:3, M:2, I:1}, {MG(C:1, M:1, I:1)}, {FT(C:1, M:2, I:1)}, {CG(C:1, M:3, I:1)}, {BT_epio (C:2, M:1, I:3)}, {BT_full(C:3, M:1, I:2)}로 특성 분석되었다.

이후 Table 4는 응용프로그램 특성 분석 단계에서 키비아트 차트로 표현되고 이 차트를 바탕으로 응용프로그램간의 자원 간섭률이 계산된다. Table 5는 16 노드에서 응용 특성

Table 4. Profiling Results by Problem Size and Number of Profiling Nodes

NPB	Single node, Class C, D			Multi nodes, Class D, E								
	1 nodes			4 nodes			8 nodes			16 nodes		
	C	M	I	C	M	I	C	M	I	C	M	I
BT	3	1	1	2	1	1	2	1	1	3	1	1
EP	3	1	1	1	1	1	2	1	1	2	1	1
LU	3	1	1	2	2	1	2	1	1	2	1	1
SP	3	3	1	1	2	1	3	1	1	3	1	1
IS	1	2	1	1	2	1	3	2	1	2	2	1
MG	1	2	1	1	2	1	1	1	1	2	2	1
FT	2	3	1	1	2	1	1	2	1	1	2	1
CG	2	3	1	1	3	1	1	3	1	2	3	1
BT_epio	3	1	3	1	1	3	2	1	3	3	1	3
BT_full	3	1	2	3	1	2	3	1	2	3	1	2

Table 5. Interference Ratio Generated as a Result of Profiling to 16 Nodes

	BT, SP	SP, LU	IS, MG	FT	CG	BT_ epio	BT_ full
BT, SP	3.027	2.161	2.379	1.472	2.408	3.027	3.027
SP, LU	2.161	2.161	2.161	1.443	2.161	2.161	2.161
IS, MG	2.379	2.161	3.462	2.161	3.462	2.553	2.524
FT	1.472	1.443	2.161	2.161	2.161	1.646	1.617
CG	2.408	2.161	3.462	2.161	4.761	2.627	2.584
BT_epio	3.027	2.161	2.553	1.646	2.627	6.490	4.761
BT_full	3.027	2.161	2.524	1.617	2.584	4.761	4.761

분석 결과로 도출된 응용프로그램간 자원 간섭률로, 마찬가지로 단일노드, 4노드, 8노드의 경우도 자원 간섭률을 계산하여 4.4장 스케줄링 최적화 기법의 확장성 실험에서 노드 수에 따른 프로파일링 기법의 스케줄링 최적화 성능 분석시에 사용하였다.

4.3 스케줄링 최적화 기법의 확장성 실험

Table 4의 응용프로그램 특성 분석 결과를 바탕으로 작업 스케줄링 실험을 통해 최적화 기법의 확장성을 분석하고자 한다. 스케줄링 실험은 랜덤으로 생성된 NPB 작업 30개가 대기중인 작업큐를 대상으로, 두개의 Worker가 스케줄링 알고리즘에 따라 작업을 가져와서 실행한다. 각 Worker는 클러스터 노드에 걸쳐서 mpi로 작업을 분산하여 실행한다.

선행 연구에서 개발한 응용프로그램 특성 분석 기반 자원 간섭률 고려 스케줄링 최적화 기법(InterAW)은 간섭률이 낮은 작업을 동시에 우선 스케줄링 하는 알고리즘[11]이며 이는 Table 6과 같다. 알고리즘은 큐에 대기중인 i개의 작업 리스트를 입력값으로 동작하며 각 작업은 스케줄링 되지 않았음을 의미하는 상태 0으로 초기화 되어 있다. 작업큐의 앞에서부터 순서대로 각 작업의 상태를 의미하는 TStatus[k]의 값이 1이 아닌 작업을 대상으로 알고리즘을 수행한다. 첫 번째 작업(QTask[0])의 경우를 보면, 작업의 상태가 1이 아니므로 α에 할당되고, 첫 번째 작업을 제외한 그 뒤의 작업들(QTask[k:i]중에 α에 할당된 작업과 간섭률이 가장 낮은 작업을 찾아 β에 할당한다. 작업 α와 β는 각각 worker1과 worker2에 스케줄링 되어 실행되며 작업의 상태는 1로 바뀐다. 이후 남은 작업들에 대해서도 같은 방식으로 진행되는데 worker1과 worker2중에 먼저 작업이 끝난 worker가 작업을 실행하고 있는 worker와 비교해서 간섭률이 낮은 작업을 선택해서 실행한다.

Table 6. Resource Interference-aware Co-scheduling Algorithm[11]

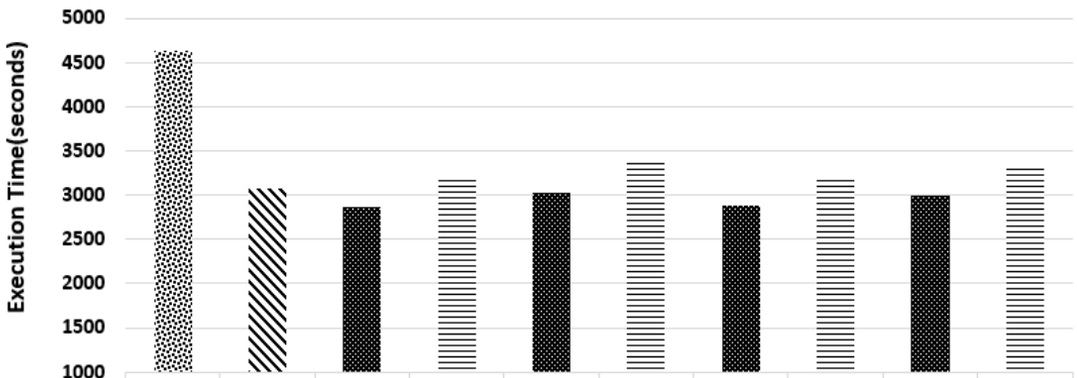
Input: Waiting job list QTask[0:i]

```
1: Set TStatus[i] = 0 ;
2: for k := 0 to i-1 step 1 do
3:     if TStatus[k] == 1 then
4:       break;
5:     else
6:         α <- QTask[k];
7:         β <- findLeastInterRatio(α, QTask[k:i]);
8:         Schedule α, β to Worker1, Worker2;
9:         TStatus[α], TStatus[β] = 1;
       else
10:    end if
11: end for
```

Output: Scheduling decision SD
= { (α, β) | α, β ∈QTask[0:i] }

본 실험에서는 제시한 InterAW 스케줄링 알고리즘과 비교군으로 선입선출 스케줄링(FCFS) 기법과 작업 실행시간이 짧은 작업 우선 실행 스케줄링(SJF) 기법을 비교 실험하였다. 또한 자원 간섭 없이 하나의 worker로 작업을 하나씩 순차적으로 실행한 결과는 'SEQ'으로 나타내었다. 3장에서 소개한 것처럼 슈퍼컴퓨터 5호기는 다른 사용자의 작업과 동시 실행시 성능 하락을 피하기 위해 노드 하나당 작업 하나를 할당하는 배타적 정책을 펼치기 때문에 SEQ 스케줄링 알고리즘에 적용하여 생각할 수 있다.

각 스케줄링 기법은 동일한 작업큐를 사용하여 비교 실험하였으며 Fig. 4는 다섯 개의 작업 큐의 평균 실행 시간으로 계산한 결과를 보여준다. 결과를 보면, 16개 노드를 전부 사용한 경우 FCFS, SJF, SEQ 대비 7.00%, 10.91%, 38.34%의 성능이 향상되었다. 또한 30개의 작업을 실행하는데 있어서 2858.24초로 전체 스케줄링 기법과 비교하여도 가장 낮은 실행시간을 보였다. 마찬가지로 동일한 응용프로그램을 사용하면서 노드 수만 반으로 줄인 8노드 프로파일링 결과에서도 FCFS, SJF, SEQ 대비 1.49%, 11.16%, 34.69%의 성능 향상을 보였다. 노드 수를 1/4로 줄인 4노드 실험의 경우 스케



	SEQ	FCFS	16-node InterAW	16-node SJF	8-node InterAW	8-node SJF	4-node InterAW	4-node SJF	1-node InterAW	1-node SJF
Execution Time	4635.84	3073.62	2858.24	3208.38	3027.64	3408.04	2889.36	3176.61	2990.11	3304.12

Fig. 4. Experiment Results of Scheduling Optimization

줄링 기법 대비 5.99%, 9.04%, 37.67%의 성능 향상을 보였으며 평균 실행시간도 2889.36초로 16노드대비 겨우 1.08% 늘어난 것을 볼 수 있다.

한편, 문제크기를 줄이고 단일 노드에서 실행한 경우에 16-node InterAW의 성능 대비 실행시간이 4.35%의 증가하였으나 InterAW 기법이 FCFS, SJF, SEQ 대비 2.71%, 9.50%, 35.50%로 실행 시간을 단축 시켜 스케줄링 기법의 최적화 성능 향상을 동일하게 나타났음을 알 수 있다. 특히 SEQ 기법 대비 노드 수별 InterAW기법은 34%~38%의 실행 최적화를 보였는데 이는 배타적으로 작업별 노드를 할당하는 슈퍼컴퓨터의 스케줄링 방식이 하드웨어 성능 카운터 기반 프로파일링을 통해 시스템 최적화 가능함을 보여준다.

5. 결 론

본 연구에서는 자체 개발한 KISTI-GVP 시스템 기반 클러스터 환경에서 응용 특성 분석 기법의 스케줄링 최적화 확장성 실험을 진행하였다. 실험 결과 프로파일링 대상 응용과 스케줄링 최적화 대상 응용프로그램의 문제 크기가 다를지라도 시스템 성능 향상의 효과가 유지됨을 보였다. 이는 슈퍼컴퓨터와 같은 대규모 노드로 구성된 클러스터 시스템의 시스템 최적화를 위한 프로파일링 테스트베드를 운용하는 것이 유용함을 보여준다. 또한 프로파일링시의 시간적 비용과 프로파일링 기반 최적화 성능 향상 효과를 함께 고려하면 노드 수를 줄이거나 응용프로그램의 크기를 줄여서 프로파일링 하는 것이 응용프로그램 특성 분석에 드는 비용을 줄일 수 있음을 확인하였다.

향후, 응용프로그램의 런타임 전체를 프로파일링 하는 것이 아닌 작업의 일부분만을 프로파일링 하는 경우의 성능 변화를 분석할 예정이다. 또한 쉘 프로그램을 이용하여 리눅스 perf 프로그램을 사용하여 mpi로 작성된 응용프로그램의 성능이벤트를 프로세스 레벨로 수집하는 데에 소모되는 오버헤드를 낮추고 데이터의 정확도를 유지할 수 있는 방법을 연구하고자 한다. 이와 같은 추가적인 연구를 통해 현재 실험에 사용한 자원 간섭률 기반 스케줄링 알고리즘을 개선하고 기초적인 스케줄링 기법이 아닌 최신 연구들에서 제시된 성능이 뛰어난 스케줄링 알고리즘과 비교 가능하도록 발전시키고자 한다.

References

[1] Top500 [Internet], https://www.top500.org

[2] B. Yang, X. Ji, X. Ma, X. Wang, T. Zhang, X. Zhu, N. El-Sayed, H. Lan, Y. Yang, J. Zhai, W. Liu, and W. Xue, “End-to-end I/O Monitoring on a Leading Supercomputer,” in *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation*, Boston, MA, USA, pp.379-394, 2019.

[3] X. Ji, B. Yang, T. Zhang, X. Ma, X. Zhu, X. Wang, N. El-Sayed, J. Zhai, W. Liu, and W. Xue, “Automatic, Application-Aware I/O Forwarding Resource Allocation,” in *Proceedings of the 17th USENIX Conference on File and Storage Technologies*, Boston, MA, USA, pp.265-279, 2019.

[4] S. Chunduri, S. Parker, P. Balaji, K. Harms, and K. Kumaran, “Characterization of MPI Usage on a Production Supercomputer,” in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Dallas, TX, USA, pp.386-400, 2018.

[5] P. Gomez-Sanchez, D. Encinas, J. Panadero, A. Bezerra, S. Mendez, M. Naiouf, A. D. Giusti, D. Rexachs, and E. Luque, “Using AWS EC2 as Test-Bed infrastructure in the I/O system configuration for HPC applications,” *Journal of Computer Science & Technology*, Vol.16, No.2, pp.65-75, 2016.

[6] B. H. Park, S. Hukerikar, R. Adamson, and C. Engelmann, "Big Data Meets HPC Log Analytics: Scalable Approach to Understanding Systems at Extreme Scale," in *2017 IEEE International Conference on Cluster Computing*, Honolulu, HI, USA, pp.758-765, 2017.
[7] S. Oral, S. S. Vazhkudai, F. Wang, C. Zimmer, C. Brumgard, J. Hanley, G. Markomanolis, R. Miller, D. Leverman, S. Atchley, and V. V. Larrea, "End-to-end I/O Portforlio for the Summit Supercomputing Ecosystem," in *SC'19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, USA, pp.1-14, 2019.
[8] G. Wei, H. Yang, Z. Luan, and D. Qian, "iDPL: A Scalable and Flexible Inter-continental Testbed for Data Placement Research and Experiment," in *2017 IEEE Symposium on Computers and Communications*, Heraklion, Greece, pp.1158-1163, 2017.
[9] S. Ilager, R. Muralidhar, K. Rammohanrao, and R. Buyya, "A Data-Driven Frequency Scaling Approach for Deadline-aware Energy Efficient Scheduling on Graphics Processing Units (GPUs), in *Proceedings of the 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet*, Melbourne, Australia, pp.1-10, 2020.
[10] S. Wallace, X. Yang, V. Vishwanath, W. E. Allcock, S. Coghlan, M. E. Papka, and Z. Lan, "A Data Driven Scheduling Approach for Power Management on HPC Systems," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, Utah, USA, pp.656-666, 2016.
[11] T. Bridi, "Scalable Optimization-based Scheduling Approaches for HPC Facilities," PhD. Dissertation, University of Bologna, Italy, 2018.
[12] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of Overprovisioned HPC Data Centers Under a strict Power Budget," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New Orleans, LA, USA, pp.807-818, 2014.
[13] A. J. Younge, R. E. Grant, J. H. Laros Ⅲ, M. Levenhagen, S. L. Olivier, K. Pedretti, and L. Ward, "Small Scale to Extreme: Methods for Characterizing Energy Efficiency in Supercomputing Applications," *The Sustainable Computing: Informatics and Systems*, Vol.21, pp.90-102, 2019.
[14] Perf [Internet], https://perf.wiki.kernel.org/index.php/Main_Page
[15] Oprofile [Internet], https://oprofile.sourceforge.io/about/
[16] Papi [Internet], http://icl.cs.utk.edu/papi/index.html
[17] Intel vtune [Internet], https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/analyze-performance/hardware-event-based-sampling-collection.html
[18] AMD uProf [Internet], https://developer.amd.com/wordpress/media/2013/12/User_Guide.pdf
[19] IBM HPCS Toolkit [Internet], https://researcher.watson.ibm .com/researcher/files/us-hfwen/HPCST_README.pdf
[20] J. Choi, G. Park, and D. Nam, "Interference-aware co-scheduling method based on classification of application characteristics from hardware performance counter using data mining," *The Cluster Computing*, Vol.23, pp.57-69, 2020.
[21] PBS Scheduler [Internet], https://www.pbspro.org
[22] Slurm Scheduler [Internet], https://www.slurm.schedmd.com
[23] Nurion [Internet], https://www.ksc.re.kr/gsjw/jcs/hd
[24] Nas Parallel Benchmarks [Internet], https://www.nas.nasa.gov/publiccations/npb.html.

최 지 은
https://orcid.org/0000-0001-6062-1376
e-mail : jieun1205@kisti.re.kr
2014년 숙명여자대학교 컴퓨터과학과 (이학사)
2016년 숙명여자대학교 컴퓨터과학과 (이학석사)
2017년 ~ 현 재 한국과학기술정보연구원 국가슈퍼컴퓨팅본부 슈퍼컴퓨터기술개발센터 기술원
관심분야 : 분산시스템, 고성능 컴퓨팅, 메타 스케줄링, 시스템 프로파일링

박 근 철
https://orcid.org/0000-0001-8388-9471
e-mail : gcpark@kisti.re.kr
1998년 중앙대학교 컴퓨터공학과(공학사)
2000년 중앙대학교 컴퓨터공학과 (공학석사)
1998년 ~ 2000년 ㈜넥센
2000년 ~ 2005년 ㈜창성정보시스템
2006년 ~ 현 재 한국과학기술정보연구원 국가슈퍼컴퓨팅본부 슈퍼컴퓨터기술개발센터 책임연구원
관심분야 : 분산컴퓨팅, 프로파일링, 작업최적화

노 승 우

https://orcid.org/0000-0001-6592-5053
e-mail : seungwoo0926@kisti.re.kr
2009년 서울시립대학교
전자전기컴퓨터공학부(공학사)
2011년 서울시립대학교
전자전기컴퓨터공학과(공학석사)
2011년 ~ 현 재 한국과학기술정보연구원 국가슈퍼컴퓨팅본부
슈퍼컴퓨터기술개발센터 기술원
관심분야 : 분산컴퓨팅, 고성능컴퓨팅, 벤치마크, 시스템
프로파일링

박 찬 열

https://orcid.org/0000-0001-6508-6114
e-mail : chan@kisti.re.kr
1993년 고려대학교 수학과(이학사)
1995년 고려대학교 전산학과(이학석사)
2000년 고려대학교 전산학과(이학박사)
1999년 ~ 2002년 ㈜이씨오
2010년 ~ 2011년 뉴욕주립대 객원연구원
2002년 ~ 현 재 한국과학기술정보연구원 국가슈퍼컴퓨팅본부
슈퍼컴퓨터기술개발센터장
관심분야 : 초고성능컴퓨팅, 분산컴퓨팅, 성능최적화, 프로비저닝