PERQ: Fair and Efficient Power Management of Power-Constrained Large-Scale Computing Systems

# Abstract

Tirthak Patel [patel.ti@husky.neu.edu](mailto:patel.ti@husky.neu.edu) Northeastern University

Devesh Tiwari [tiwari@northeastern.edu](mailto:tiwari@northeastern.edu) Northeastern University

**Table 1:** Applications from Exascale Computing Project Proxy App

대규모 컴퓨팅 시스템은 점점 더 많은 전력 제약을 받고 있지만, 이러한 시스템은 애플리케이션 양이온이 노드의 피크 전력 용량을 소모하지 않기 때문에 더 높은 시스템 처리량을 달성하기 위해 하드웨어 오버 프로비저닝을 사용한다. 불행히도, 시스템 처리량에만 집중하면 여러 번 동시에 실행되는 응용 프로그램 중 심각한 불공정성을 초래할 수 있습니다. 본 논문에서는 동시 응용 프로그램 간의 공정성을 달성하면서 시스템 처리량을 향상시키기 위한 새로운 피드백 기반 원칙적 접근 방식인 PERQ를 소개한다.

**ACM Reference Format:**

Tirthak Patel and Devesh Tiwari. 2019. PERQ: Fair and Efficient Power Management of Power-Constrained Large-Scale Computing Systems. In *The 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC ’19), June 22–29, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, [12](#_bookmark30) pages. <https://doi.org/10.1145/3307681.3326607>

# 1. Introduction

지속적인 진보의 고성능 컴퓨팅은 계산 과학자들이 과학적 발견을 촉진할 수 있게 해 주었지만, 대규모 시스템의 높은 전력 소비는 미래 규모 시스템의 상위 10개 과제 중 하나이다. 현대 데이터 센터의 대규모 엔터프라이즈 컴퓨팅 시스템도 여전히 전력 제약을 받고 있다. 이러한 제한된 전력 가용성은 높은 시스템 처리량을 얻기 위해 지능형 작업 스케줄링 방법이 필요하다.사전 연구들은 하드웨어 과다 프로비저닝이 전력 제약 대규모 시스템-템의 효율성을 증가시키는데 효과적일 수 있다는 것을 보여주었다. 이 접근법에 따르면, 각 노드가 항상 노드의 최대 전력 수준에서 작동하는 경우 시스템 전력 예산이 수용할 수 있는 것보다 더 많은 수의 계산 노드로 대규모 시스템이 제공됩니다 [[45,](#_bookmark72) [55].](#_bookmark82)

**Why over-provision large-scale computing systems?** 지나치게 프로비저닝하는 이론은 핵심 통찰력에 의존합니다. 응용 프로그램은 Table 1에서 볼 수 있듯이 컴퓨팅 노드의 지정 열 설계력 (TDP) 한계보다 노드 당 낮은 전력을 일반적으로 소비합니다. (다중 HPC ap-plication의 노드당 평균 소비 전력은 Intel Xeon E5-2686 노드에서 TDP의 25%-70% 범위에 있습니다). 전통적으로 시스템 설계자는 최악의 경우 제공이라고 불리는 주어진 시스템 전력 예산에 따라 최대 용량(TDP)으로 전원을 공급할 수 있는 만큼 시스템의 노드를 공급할 것이다.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [p](mailto:permissions@acm.org)[ermissions@acm.org.](mailto:ermissions@acm.org)

*HPDC ’19, June 22–29, 2019, Phoenix, AZ, USA*

© 2019 Association for Computing Machinery. ACM ISBN 978-1-4503-6670-0/19/06. . . $15.00

<https://doi.org/10.1145/3307681.3326607>

Suite [[1],](#_bookmark27) used for experiments described in Sec. [3.](#_bookmark18)

|  |  |  |
| --- | --- | --- |
| **Application** | **Domain** | **Avg. Power (% of TDP)** |
| ASPA | Multi-scale physics | 27% |
| CoHMM | Material shockwave analysis | 27% |
| CoMD | Molecular dynamics | 48% |
| HPCCG | Conjugate gradient proxy | 57% |
| RSBench | Multipole resonance | 39% |
| SimpleMOC | 3D neutron transport in reactor | 69% |
| budget SWFFT | Cosmology | 28% |
| XSBench | Monte Carlo neutronics | 43% |
| miniFE | Unstructured finite element solver | 61% |
| miniMD | Parallel molecular dynamics | 65% |

오버 프로비저닝은 시스템이 최악의 경우 프로비저닝된 시스템에 비해 더 많은 수의 작업을 동시에 실행할 수 있도록 하여 시스템 작업 처리량(즉, 단위 시간당 완료된 작업)을 증가시킨다. 이러한 기회를 인식한 연구원들은 오버 프로비저닝 시스템을 보다 효율적으로 만드는 데 상당한 진전을 이루었습니다 [19, 20, 45, 46, 54, 55, 59]. 선행 연구들은 과잉 프로비저닝된 대규모 클러스터의 경제적 생존 가능성을 입증했으며[47], Production scale에서의 과잉 프로비저닝 시스템이 실질적으로 실현 가능하고 유익하다는 것을 뒷받침하는 실험적 증거를 제공했다. 엔터프라이즈 컴퓨팅 데이터 센터는 피크로드 요구를 충족시키기 위해 하드웨어 리소스를 과다 프로비저닝하는 경향이 있습니다 [8, 15, 44].

**Need for balancing system throughput and fairness on power-constrained hardware over-provisioned system:** 지나치게 프로비저닝된 시스템에서 시스템 처리량을 늘리는 직관적인 접근법은 모든 노드에서 작업을 실행하지만 동일한 수준에서 모든 노드를 전원 공급합니다. 이러한 접근방식은 기본 시스템(최악의 경우 프로비저닝 시스템)보다 노드 수를 더 많이 사용할 수 있어 작업 처리량이 높아질 가능성이 높기 때문에 유망하다. 동시에 이 접근법은 모든 계산 노드에 전원이 고르게 분배되기 때문에 다른 작업에도 "공정"합니다. 그러나, 선행 연구들은 그러한 "공정성 지향 정책"이 과잉 제공된 시스템의 시스템 처리량에 충분한 개선을 가져오지 않는다는 것을 보여주었다. [[19](#_bookmark46), [33](#_bookmark60), [42](#_bookmark69), [46](#_bookmark73)]. 기본적으로 ‘공정성 지향 정책’은 시스템 처리량을 향상시킬 수 있지만, 과잉공급의 자본과 운영 비용을 극복하기에 충분하지는 않다. – 과공급 인자(즉, 본 논문에서 f라고 하는 최악의 경우 전력 프로비저닝 시스템의 노드 수에 대한 과공급 시스템의 노드 총 수의 비율)에 의해 결정된다.

따라서, 연구자들은 지나치게 공급된 시스템의 자본과 운영 비용을 능가하기 위해, 스케줄러가 시스템 처리량을 향상시킬 가능성이 있는 일자리에 더 많은 전력을 할당하는 데 초점을 맞출 수 있는 "처리량 지향 정책"을 설계했다. 예시적으로 직관적인 접근방식은 마무리에 가장 가까운 작업에 최대의 힘을 부여하고 최소의 노드에서 실행되는 것이다. 시스템 처리량을 향상시키는 데 효과적이지만, 그러한 정책은 근본적으로 설계에 의해 불공평하다(최악의 경우 우리의 결과가 보여주듯이,

그러한 접근방식은 공정한 성능 수준에서 일부 애플리케이션에 대해 200% 이상의 성능 저하를 초래할 수 있다. (Sec).3) 우리는 대규모 HPC 시스템이 주로 고성능을 위해 설계되었지만 공정성 속성을 자원 관리 정책에 통합하지 못하면 바람직하지 않은 부작용(예: 과학적 발견의 의도하지 않은 지연, 부정확한 자원 소비 회계, 사용자의 불공정한 처리)이 발생할 수 있다는 점에 주목한다.

요약하자면, (1) 과잉공급 비용을 보완하기 위해 더 높은 시스템 처리량을 달성하고 (2) 직무간 공정성을 유지할 수 있는 두 가지 상충되는 목표를 동시에 충족시킬 수 있는 전력관리 프레임워크가 필요하다. 규칙 기반 또는 Ad-hoc 기반 전력 관리 전략은 그러한 접근법이 동적 피드백과 보증을 제공하기 위한 이론적 토대가 부족하기 때문에 두 목표를 동시에 달성할 수 없습니다. 기계 학습 기반 기술은 응용 프로그램의 동작을 배우고 능동적으로 전력 할당을 조정할 수 있지만 여러 충돌 목표를 최적화할 때 이론적 보장을 제공할 수는 없습니다.

**We propose PERQ** 1**, a control-theoretic policy to allocate power in a way which achieves high system throughput, while being fair to the jobs.** PERQ는 이 두 가지 목표를 동시에 달성하기 위해 다중 객관적 제어 이론의 원리를 사용합니다. PERQ는 동적 피드백을 사용하여 다양한 특성을 가진 작업에 적응하고 파워 캡핑 결정의 성능 영향을 평가합니다. PERQ가 이러한 목표를 달성할 수 있게 하는 HPC 애플리케이션의 특징은 Sec2에서 논의됩니다.

**Contributions:** 본 논문에서는 PERQ를 소개하고, 시스템 처리량을 개선하여 과다 공급 비용을 보완하고, 동시에 운영되는 업무들 간의 공정성을 유지할 수 있도록 하는 새로운 원칙적인 전력 할당 기법이다. 공정성을 제공하고 시스템 처리량을 극대화하기 위해 PERQ는 다른 작업의 성능을 해치지 않고 시스템 처리량에 큰 영향을 미치는 작업에 더 많은 전력을 제공합니다. 이 기회는 애플리케이션이 파워 캡에 대해 다른 수준의 민감도를 가지고 있다는 관찰을 활용함으로써 가능합니다. 일부 작업은 더 높은 파워 캡에서와 마찬가지로 낮은 파워 캡에서 똑같이 잘 수행되지만 다른 작업은 더 민감합니다. PERQ는 최적 및 견고한 제어 이론을 사용하여 특정 작업의 전원 캡을 신중하게 줄이고 시스템 처리량을 극대화하기 위해 다른 작업의 전원 할당을 증가시킵니다. 그러나 이러한 결정을 내리려면 서로 다른 (보이지 않는) 일자리에 대한 전력 할당 수준과 성능 관계를 정확하게 추정해야 한다. – 공정하고 효율적인 전력 관리를 제공하는 데 가장 어려운 과제입니다. 이러한 과제를 극복하기 위해 PERQ는 시스템 식별 이론을 사용하여 유도된 새로운 상태 공간 시스템 모델을 구축합니다 [34].이 모델은 워크로드에 모델을 과대 적합시키지 않고 전력 할당 수준과 성능 관계를 정확하게 추정합니다. 이전에 보이지 않았던 응용 프로그램 세트를 사용하여 평가한 결과 PERQ는 원하는 목표 레벨을 충족하고 성능 향상을 가져오지만 PERQ는 완전히 다른 벤치 마크 세트를 사용하여 시스템 모델을 구축합니다. 전체 PERQ 실험, 측정 데이터 및 프로토 타입 구현은 <https://github.com/GoodwillComputingLab/PERQ> 에서 연구 커뮤니티를 위한 오픈 소스로 제공됩니다.

실험 및 시뮬레이션을 통한 평가 결과 PERQ는 다른 직업의 파워 캡 민감도의 차이를 활용하여 과도하게 공급된 시스템에 대한 시스템 처리량을 높이고 동시에 작업을 실행하는 데 공정합니다.

1 PERQ is a conveniently chosen acronym for power provisioning for efficiency and fair equality, and is pronounced as *perk* (an additional benefit or advantage).

PERQ의 평가는 실제 대규모 HPC 시스템과 일자리의 특성에 의해 좌우된다. PERQ는 서로 다른 특성을 가진 직업에 적응적이고 안정적이고 공정한 대우를 제공한다. PERQ는 다른 수준의 과잉 프로비저닝을 하는 시스템에 대해 더 나은 시스템 처리량을 제공한다. 전반적으로 PERQ는 공정성 지향 할당 정책에 비해 시스템 처리량을 최대 50% 포인트까지 향상 시키며 공정하게 유지합니다.PERQ는 HPC 시스템이 과도한 프로비저닝의 자본 및 운영 비용에 비해 더 높은 이익을 얻을 수 있도록 도와줍니다.

**PERQ makes the state-of-art for data center power man- agement landscape richer and easier to advance:** 과다 제공된 대규모 클러스터가 PERQ의 주요 목표 시스템이지만 PERQ 솔루션의 핵심 기술 혁신은 엔터프라이즈 데이터 센터의 전력 관리에도 적용 가능하고 유용합니다. 데이터 중심의 전력 관리는 데이터 센터 운영을 발전시키고 개선하는 데 중심이 되어 왔다.연구자들은 전력 예산 위반을 최소화하고 전력 진동 관계를 줄이고 효율성을 향상시키는 등 서로 다른 목표를 달성하기 위해 피드백 기반, 다중 레벨, 조정된 접근법을 사용하는 지능형 전력 할당 기술을 설계했습니다 [[4](#_bookmark31), [15](#_bookmark42), [28](#_bookmark55), [32](#_bookmark59), [40](#_bookmark67), [48](#_bookmark75), [51](#_bookmark78), [58](#_bookmark85), [60](#_bookmark87)]. 그러나, Li et al.에 의해 언급된 바와 같이. HPCA2019 [31]에 발표된 최근 연구에서, 우리의 작업과 동시에, 현재의 최신 최신 데이터 센터 전력 관리 체계는 작업량 우선 순위를 무시하고 있으므로 공정성을 유지할 수 없다. Li et al.[31] 데이터 센터 파워를 관리하기 위해 첫 번째 워크로드 우선 순위 인식 확장 가능한 접근법을 설계하는 데 상당한 진전을 보였지만 PERQ와 같이 여러 충돌하는 목표, 수렴 및 안정성을 충족시키는 데 대한 이론적 보장을 제공하지 않습니다. 데이터 센터 전력 관리 프레임 워크에 PERQ를 통합하면 중요한 격차가 해소됩니다. 동적으로 변화하는 환경에서 안정성과 수렴에 대한 입증 가능한 보장으로 공정성과 효율성 목표를 공동으로 달성합니다. 또한, 연구자들은 PERQ의 원칙적인 다목적 제어-이론적인 접근을 이용하여 데이터 센터 전력 관리 체계를 설계할 수 있다. 우리의 오픈 소스 PERQ 모델 예측 제어기 모듈은 서로 다른 목표와 시스템에 대한 새로운 데이터 센터 전력 관리 기술을 설계하기 위한 진입 장벽을 낮춘다.

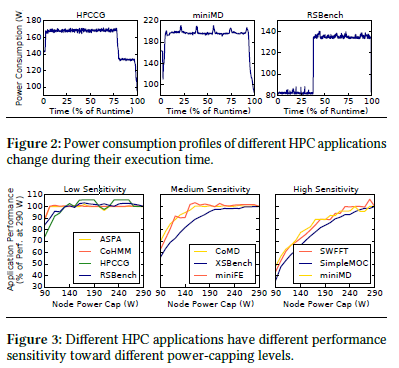
# 2. PERQ Framework

PERQ는 (1) 과잉 프로비저닝(시스템의 목표) 비용을 보상하기 위해 높은 시스템 처리량을 달성하고 (2) 동시 실행되는 모든 작업(직무의 목표)에 공정하게 유지하는 두 가지 상충되는 목표를 관리하기 위해 다목적 제어 이론을 활용한다. PERQ는 동적 피드백을 사용하여 개별 작업의 성능에 대해 배우고 작업 및 시스템 수준 목표가 모두 충족되도록 개별 노드를 캡핑하여 전력 할당을 조정합니다. PERQ는 작업 및 시스템 목표에서 수렴 및 안정 상태에 대한 이론적 보장을 제공합니다. 다음으로, 피드백 기반 솔루션의 필요성과 타당성을 입증하기 위해 HPC 애플리케이션의 특성과 PERQ가 충돌하는 목표를 효율적으로 충족시키기 위해 이러한 특성을 어떻게 활용하는지에 대해 논의한다.

# Enabling HPC Applications Characteristics

**Observation 1: HPC jobs are typically long-running and pro- vide opportunity for feedback-based dynamic solution.**

피드백 기반 동적 솔루션이 효과적이기 위해서는 작업을 충분히 오래 실행하여 변경하고 효과를 관찰해야 합니다.예를 들어, 이 일이 몇 밀리초 동안만 지속된다면,



1.0

Mira Trinity

0.8

0.6

CDF

0.4

0.2

0.0

0 5 10 15 20

Runtime (hr)

**Figure 1:** HPC applications are sufficiently long-running to be

suitable for a feedback based control.

파워 캡핑을 적용하고 파워 캡핑 수준을 재조정하기 위해 적시에 그 효과를 관찰하는 것은 어려울 것이다.왜냐하면 파워 캡핑 자체는 몇 밀리초가 걸릴 수 있다. 다행히도 HPC 작업은 일반적으로 몇 분에서 몇 시간, 심지어 며칠 동안 실행됩니다. 그림에서 보듯이. 1, HPC 작업은 LANL의 Trinity [14]와 Argonne의 Mira [27] 슈퍼 컴퓨터 등 여러 슈퍼 컴퓨터에서 충분히 길다. 평균 근무 시간은 트리니티에서 30분, 미라에서 72분이다.미라에 대한 일자리의 62% 이상과 트리니티에 대한 일자리의 46%가 30분 이상 런타임을 가지고 있다. 이러한 특성으로 PERQ는 제어(전원 캡핑)를 적용하고 일생 동안 효과(수행)를 관찰할 수 있는 피드백 기반 솔루션을 쉽게 적용할 수 있다.

**Observation 2: HPC applications may exhibit dynamic power consumption behavior during their execution; making static or ad-hoc power allocation not suitable and requiring adaptive solution like PERQ.** 일반적으로 HPC 애플리케이션은 위상 기반 동작으로 인해 실행 중에 단조로운 자원 활용을 하지 못합니다 [18,26, 30]. 결과적으로 실행 시간 동안 다양한 전력 소비를 나타냅니다. 이걸 지지해주려고, 그림. 2는 런타임 동안 표 1에서 나온 3개의 애플리케이션의 전력 소비 프로파일을 보여준다(다른 애플리케이션은 유사한 동작을 나타낸다). 따라서 작업 시작 시 하나의 전원 캡을 설정하고 실행 중에 동일한 전원 캡을 유지하는 정적 솔루션을 사용할 수 없음을 보여줍니다. 이것이 PERQ가 이러한 단계에 적응하고 그에 따라 전원 캡을 변경하기 위해 동적 피드백을 사용하는 이유입니다. 이러한 단계는 종종 충분히 긴 기간이며 매우 자주 변경되지 않습니다 [18, 43]. HPC 응용 프로그램은 종종 반복적이며 유사한 계산을 반복적으로 수행하기 때문입니다. 이를 통해 PERQ의 컨트롤러는 안정된 솔루션으로 조정하고 수렴할 수 있으며 위상이 변경되면 적절하게 반응할 수 있습니다.

**Observation 3: HPC applications have varied sensitivity to- ward power-capping.** PERQ는 HPC 응용 프로그램의 흥미로운 특성을 활용합니다. 파워 캡핑은 HPC 응용 프로그램의 성능에 다른 파워 캡핑 수준에서 다른 수준으로 영향을 미칩니다. 그림에서 보듯이.3) HPC 응용에 대한 파워 캡핑의 효과는 세 가지 범주로 분류할 수 있다. (1) low-sensitivity,

(2) medium-sensitivity, and (3) high-sensitivity.

파워 캡에 대한 민감도가 낮은 응용 프로그램은 파워 캡 레벨이 TDP보다 현저히 감소함에 따라 극적인 성능 저하를 관찰하지 않습니다. 예를 들어, at90W 파워캡 레벨(TDP는 290W와 동일함)에서도 ASPA, CoHMM, HPCG와 같은 애플리케이션은 최고 파워캡(290W)에 비해 20% 미만의 성능 저하를 관찰한다. 이는 이러한 애플리케이션들이 주로 메모리 집중 또는 통신 집중이기 때문이다. 반면에 파워 캡에 대한 높은 민감도를 가진 응용 프로그램은 파워 캡 레벨이 떨어지면서 가파른 곡선으로 극적인 성능 저하를 관찰하며 60% 이상의 성능 저하를 관찰합니다.

파워 캡핑에 대한 중간 민감도를 가진 응용 프로그램은 매우 가파르거나 평평한 파워 캡핑 성능 곡선이 아닌 중간 동작을 나타냅니다.우리는 응용 프로그램에서 사용되는 활성 코어의 수가 파워 캡에 대한 민감도의 절대 지표가 아니라는 것을 발견했습니다. For instance, both, SimpleMOC and miniFE use all of the available CPU cores; however, Simple- MOC is much more sensitive to power-capping, while miniFE is less sensitive than even applications which run on only 32 cores such as XSBench and miniMD. In fact, miniFE’s performance curve is similar to that of CoMD, which utilizes only 16 cores. Moreover, an application’s sensitivity also changes according to the phase it is in (i.e., compute-intensive phase vs. memory-intensive phase). Due to these complexities in performance impact of power-capping, it is non-trivial to use any ad-hoc policies to meet the two goals.

As described next, PERQ exploits this observation to intelligently re-adjust and shuffle power allocation among jobs such that it remains fair to different jobs while maximizing system throughput.

# PERQ Approach

PERQ uses the principles of multi-objective control theory to achieve both system- and job- level objectives simultaneously. The system-level objective is to achieve proportionally higher through- put than it would have achieved with a worst case provisioned sys- tem. The job-level objective is to achieve the performance it would have achieved had the power been distributed equally among the nodes. Note that these targets are not static; they change over time. System-level target varies according to the jobs running on the sys- tem, while job-level target varies according to the job’s individual performance and phase behavior. PERQ detects the change in the target and re-adjusts itself.

One of the key features of PERQ is that it leverages the insight that different applications exhibit different levels of sensitivity to power-capping. PERQ can cap an application exhibiting low- sensitivity toward power-capping at lower level, while achieving the same performance, and thus, being fair to it. The power saved from such application can be allocated to applications which are likely to improve system throughput. *We note that PERQ does not*

*rely on precomputed application models to derive their power-cap and performance relationships. Instead, PERQ uses feedback to dy- namically adjust to jobs with diverse characteristics, and to assess the performance impact of its power-capping decisions.*

PERQ uses multi-objective control theory to meet the dynamic goals, while satisfying hard constraints of the physical system such as the minimum and maximum power-capping levels of the nodes and the system’s total power budget. Multi-objective control theory enables PERQ to individually assign weights to the system-level throughput target and the job-level performance targets. Addition- ally, PERQ uses dynamic feedback to adapt and adjust to changing situations, and make power provisioning decisions at discreet in- tervals and for discreet targets. As discussed next, grounded in the theory of multi-objective controls, PERQ enjoys formal guarantees of responsiveness, convergence, and stability.

# Multi-Objective Control

**Table 2:** Characteristics of control theory.

|  |  |
| --- | --- |
| **Feature** | **Types** |
| Feedback | Continuous, Discrete |
| Techniques | Classical, Adaptive, Stochastic, Optimal |
| Size | SISO, SIMO, MISO, MIMO |
| Model | Static, Dynamic, Uncertain, Constrained |

* + 1. **Controller Design** A control theoretic solution requires the selection and design of an appropriate controller to meet the desired goal(s). Table [2](#_bookmark6) outlines the features and system charac- teristics which categorize different controller types. Traditionally, “classical” controllers belonging to the Proportional-Integrative- Derivative (PID) family are used in many situations, as they pro- vide strong convergence and stability guarantees. However, these controllers are not suitable for PERQ since PID controllers cannot optimally control a system with multiple conflicting goals. On the other hand, controllers that can control multiple objectives and belong to the “optimal” technique family instead of “classical” (e.g., Linear-Quadratic Regulators (LQR) controllers) will not be effective for PERQ’s case. This is primarily because such controllers can only provide a one-time optimal strategy for a system (assuming a static system model for infinite time) and hence, are not capable of providing optimal strategies for dynamically changing system and job characteristics, as is the case for PERQ. Therefore, PERQ chooses a Model Predictive Controller (MPC) [[29](#_bookmark56)], from the optimal and robust branch of control theory, which can handle multiple

computing systems are complex and effect of power-cap on per- formance depends on various factors. An MPC controller can deal with systems with uncertain models and dynamic conditions. (3) An MPC controller can work at discrete intervals, which is required since power-caps take some time to take effect and nodes require a discrete interval to study the impact of the power-cap on per- formance. (4) Lastly, a “constrained” MPC controller allows PERQ to meet hard system constraints. PERQ needs to ensure that the power-cap set by the controller for each node remains within the thermal design power of the node and the overall power usage of the system remains below the system power budget.

In addition to meeting the above targets, a MPC controller can be designed to ensure that the targets are reached quickly and opti- mally. The optimality condition refers to (1) the cost of not reaching the target during the prediction horizon, and (2) the cost of making large changes to the “actuator” values from one instance to another (actuator is the component which enables the control; in the case of PERQ, this is the node power-cap). The above two costs reflect the convergence and stability of the controller. A strategy that does not reach the target is non-convergent, and would thus incur a large cost pertaining to (1). On the other hand, it is not desirable to continuously drastically change a compute node’s power-cap between two consecutive decision intervals (e.g., from 290W (TDP of an Intel Xeon E5-2686 node) to 100 W to 290W) since it may lead to highly unstable job and system behavior, and may even have adverse effect on the node-health in the long term. Thus, a strategy which is unstable should incur a large cost pertaining to (2). Therefore, the optimal strategy minimizes both of these costs. Next, we describe the mathematical formulation of how a MPC controller minimizes multiple costs.

* + 1. **Mathematical Theory of MPC Controllers** The MPC controller minimizes the cost, *J* , by solving a quadratic program- ming problem (Eq. [1)](#_bookmark9) every decision instance. It finds the actuator values which minimize the overall cost for the next instance, provid- ing a convergent and stable solution. In Eq. [1,](#_bookmark9) *M* is the prediction horizon, *N* is the number of targets, *Ti*, *j Yi*, *j* is the difference between the target value and actual value (*WT* is the corresponding weight), and *Ui*, *j*,*k* +1 *Ui*, *j*,*k* is the change in the actuator values

−

−

between instance *k* and *k* + 1 (*WU* is the corresponding weight).

Thus, the cost is proportional to the square of the differences be-

tween the targets and the actual values, and the changes in the actuator values.

. .

conflicting objectives simultaneously (multiple inputs and multiple outputs (MIMO)) and can provide optimal solution in a dynamically changing environment.

*M N*

*J* =

*i* =1 *j* =1

(*WT* (*Ti*, *j* − *Yi*, *j* )2

+ *WU* (*Ui*, *j*,*k* +1 − *Ui*, *j*,*k* )2) (1)

Model Predictive Controller (MPC) [[29](#_bookmark56)] devises an optimal con- trol strategy for a fixed time window at the current decision instance. The controller analyzes the impact of its previous decision and the new targets to reassess it’s control strategy and implements another strategy over the next time window. Use of a receding time window ensures that the control strategy employed is adaptive and adjusts itself to provide optimal solution for the targeted time window. This *time window* is referred to as the *MPC prediction horizon* in MPC literature. PERQ chooses MPC controller as it provides the follow- ing desirable properties: (1) PERQ requires the controller to have MIMO controlling ability in order to handle all targets together. MPC allows PERQ to achieve this successfully by allowing to put equal weights on the system-level and job-level objectives (and it can also be configured to assign different weights). (2) Large-scale

MPC controllers have been shown to provide convergence and stability by using the conditions of “terminal cost” and “terminal region” [[6](#_bookmark33), [9](#_bookmark36), [39](#_bookmark66)]. Terminal cost is a cost which is added to the total cost if the solution does not converge by the last stage in the prediction horizon. Typically, a large terminal cost is chosen in order to enforce convergence. On the other hand, terminal region is a set of states (a state is one permutation of all actuator values i.e., in the case of PERQ, a state is a permutation of node power-caps of all nodes in the system) which the last state in the prediction horizon must belong to in order to ensure that the solution con- verges by the last stage in the prediction horizon. A large penalty cost is placed on the controller if it does not converge to one of the states in the terminal region, therefore, enforcing convergence and stability. Typically, the terminal region is dictated by the constraints



PERQ

Target Generator

Job Statuses Job

Scheduler

Node 1

Node 2

Targets

**+**

**-**

PERQ

Controller

Power-caps

Node NOP

Performance Indicators

**Figure 4:** Conceptual overview of PERQ feedback control flow.

of the system and the optimality condition of the controller [[39](#_bookmark66)]. In the case of PERQ, the terminal region includes states which meet the power constraints, and minimize the cost sufficiently (as determined by the weight parameters). It might in some cases be infeasible to achieve a cost of zero (i.e., impossible to meet all goals simultaneously); therefore, MPC controller allows convergence to a non-zero cost state as well. However, any state which does not meet the above requirements is penalized, and therefore, not allowed. By meeting these conditions, it can be ensured that the controller converges as close as possible to the targets, and remains stable around the final values.

# PERQ Feedback Control

Fig. [4](#_bookmark10) provides an overview of PERQ dynamic feedback control loop that helps PERQ adapt to changing job behavior and sys- tem environment, while meeting the system- and job-level perfor- mance objectives. Compared to a traditional system, PERQ adds two new components: “PERQ Target Generator" and “PERQ Controller". PERQ target generator module determines the target levels: sys- tem throughput for the whole system (efficiency) and the job-level performance for each job (fairness). PERQ controller receives two inputs: (1) target for system throughput for the whole system and job-level performance for individual jobs from the target generator, and (2) the current system throughput and current performance level of different jobs from compute nodes. At each decision inter- val, PERQ controller determines the power allocation for each job (and hence, power-cap level for individual compute nodes) based on the difference between the target level and current performance level. Power-capping bounds the power consumption of compute nodes as determined by the controller for a given decision interval. The feedback process is iterative and discrete, where at the end of each decision interval, compute nodes send the job’s performance under the set power-cap for this decision interval to the PERQ target generator. Then, PERQ target generator determines the new targets based on the dynamic job behavior and system state – note that the target levels for both, the system and the individual jobs, may change over decision intervals as different jobs arrive, change phases, or finish. Then, the controller receives the difference be- tween the targets and the observed performances, and determines the power-caps to ensure accurate goal tracking (the difference between the goal and the actual performance should be as small as possible, i.e., the performance should track the goal).

PERQ target generator and controller communicate with the job scheduler and resource manager to learn the status of currently running jobs and their node allocations, whenever a new job ar- rives or finishes. At every decision interval, compute nodes provide performance data that is used to calculate the current job-level performance and system throughput by the controller and estimate the target job-level performance and system throughput by the target generator.

PERQ uses a simple, effective and easy-to-measure metric, in- structions per second (IPS), as the performance/progress indicator. Compute nodes send IPS of the MPI ranks running on the node at the end of each decision interval. Note that other performance in- dicators can be used, but PERQ prefers IPS for various reasons. IPS is available every decision interval. Traditional methods of judging progress of HPC jobs such as length of a time loop or time between MPI barriers cannot be used as performance indicators because the information is not available every decision interval. It can take a long period to assess the performance impact of a certain power-cap if the time to execute a loop iteration is long. Moreover, the length of a loop iteration may vary from one job to another job, which makes it impossible to make power provisioning decisions for all of them at the same time (i.e., end of a decision interval). We note that the target generator and the controller use the IPS of the slowest job (MPI) process to determine its performance – this avoids the misleading scenario where a process waiting on a synchronization barrier may falsely indicate fast progress for the job.

The controller uses the difference between the target IPS and the current IPS to set the power-cap for each of the nodes. The target generator aggregates the IPS of all the active jobs to estimate the current system-level throughput. We note that target generator needs projected IPS information at different power-caps to generate new system- and job-level performance targets for next decision interval. Sec. [2.4.1](#_bookmark11) and [2.4.2](#_bookmark13) discuss how a state-space system model is identified to achieve this purpose.

* + 1. **PERQ System- and Job-Performance Target Genera- tion** The target generator determines the system-level and job- level targets at each decision instance.

From the system’s perspective, the system throughput target of the over-provisioned system (*TOP* ) should be higher than the throughput of an equivalent worst case provisioned system (*TW P* ). It can be expressed, in a general form, as *TOP* = *T*improv-ratio *TW P* , where *T*improv-ratio is the factor of improvement which is expected to be proportional to the increase in the size of system. For ex- ample, a system with 20% over-provisioned nodes may expect the throughput to go up by 20%. We note that such an expectation is clearly unrealistic at very high over-provisioning factors (*f* ). Our evaluation in Sec. [3](#_bookmark18) shows that PERQ is not sensitive to the value of system throughput improvement ratio (*T*improv-ratio).

*TOP* is determined dynamically based on the set of currently run- ning jobs on the over-provisioned system, *AOP* , and their respective performances. To determine the system throughput target, PERQ

needs to know the performance of a worst case power provisioned system given a set of jobs *AW P AOP* running at highest power- cap (a worst case power provisioned system has fewer jobs than an over-provisioned system because it has fewer nodes; therefore, *AW P* contains the subset of jobs in *AOP* which can be accommo- dated by a worst case power provisioned system on FCFS basis). The performance of all jobs *AW P* is summed up to provide the sys- tem’s performance. However, it is infeasible to actually run the jobs on a worst-case provisioned system to determine *TW P* . Therefore, PERQ uses a system model, discussed next, which enables PERQ to predict the IPS of jobs in *AW P* when they are running at maximum power-cap *TDP* (Sec. [2.4.2).](#_bookmark13)

⊂

From the jobs’ perspective, the performance target for each job

is to achieve the performance that it would under equal power allocation across the nodes (i.e., fairness-oriented policy). This fair power allocation can be estimated *POP* = *TDP* × *NW P* , where *NOP*

*NOP*

and *NW P* are the number of nodes in the over-provisioned and worst case power provisioned systems, respectively. Thus, PERQ needs a system model to be able to predict the performance of

Power-cap P(k)

Node

Uncertainty D(k)

V

+ State X(k) +

IPS Y(k)

each jobs during its current phase at *TDP* and *POP* . We discuss the details of this system model in the next section (Sec. [2.4.2).](#_bookmark13)

B + +

C +

A *X(k + 1) = AX(k) + BP(k) +VD(k)*

*Y(k + 1) = CX(k) + D(k)*

* + 1. **PERQ System Model** To estimate the performance of a job



at maximum power-cap (i.e., TDP) and fair power allocation (*POP* ) in an over-provisioned system, PERQ develops a system model that captures the power-cap and performance relationship for different jobs on a given system. PERQ builds a state-space model [[49](#_bookmark76)] de- rived using system identification theory [[34](#_bookmark61)]. To ensure that the model is not an over-fit for the running workloads, PERQ uses a completely different set of benchmarks to build this model, and as our evaluation (Sec. [3)](#_bookmark18) shows, PERQ performs effectively for a different set of unseen application set.

To develop a system model that captures power-cap and per- formance relationship, one could attempt to build an analytical model for the processor but the complexity of a processor would render such an approach ineffective. For example, Intel processors supporting Running Average Power Limit (RAPL) [[12](#_bookmark39)] power-cap features have a complicated power-cap vs. performance relationship due to hardware features such as dynamic voltage and frequency scaling (DVFS), dynamic clock modulation, C-state and P-state management, and Turbo-boost. The power-cap vs. performance re- lationship is further complicated by software and job activity such as OS jitter, number of active threads, CPU and memory intensity. To address this challenge, PERQ employs a data-driven model. The performance of different NAS Parallel Benchmarks (NPBs) with different input sizes was collected for different levels of power-cap on the Intel processor used for evaluation. NPB benchmarks cover a wide range of application behaviors with different input sizes, which is used to obtain power-cap vs. performance curves of HPC appli- cations with different sensitivity toward power-capping. Results were measured by running each benchmark one hundred times and switching the power-cap frequently using a uniform distribution, to emulate a real switching environment that captures impact of power-capping for different phases of the benchmarks. PERQ, then, uses the system identification method [[34](#_bookmark61)] to develop a state-space model for the power-cap vs. performance relationship of the node. Note that, such a model is sufficient for all nodes of the same type; but it needs to be rebuilt for a different type of node. However, an HPC system does not change its node type frequently, therefore, development and use of “build-one-time-use-through-out-lifetime" model is reasonable to develop the PERQ controller.

State-space model predicts the performance impact of a particu- lar power-cap based on previous inputs (previous power-cap levels) and outputs (estimated performance vs. observed performance) [[49](#_bookmark76)].

PERQ uses a 3*rd* order state-space model that uses the previous three power-caps (*P k* 3 , *P k* 2 and *P k* 1 ) and outputs IPS at the current instance, denoted as *Y k* , based on the current power-cap *P k* at decision interval *k*. Note that it is not possible to assign a static power-cap vs. performance relationship to the system as the job performance depends not only on the system hardware features, but also its own characteristics. Moreover, jobs are also prone to showing varying performance based on the phase in which they are operating. Therefore, PERQ employs a model which takes into account the dynamic impact of power-cap on the jobs’ performance. Moreover, state-space models are linear and time-invariant allowing for low-overhead analysis.

( )

( )

( − ) ( − ) ( − )

**Figure 5:** Conceptual view of the PERQ’s state-space model of the system that captures the power-cap and performance relationship, accounting for processor complexity and uncertainties.

**Table 3:** MPC matrices referred to in Eq. [2,](#_bookmark15) Eq. [3](#_bookmark16) and Eq. [4.](#_bookmark17)

|  |  |
| --- | --- |
| *NJ* | Number of jobs running on the system |
| *NOP* | Number of nodes in the over-provisioned system |
| *M* | Prediction horizon of MPC controller |
| *T* | Vector of performance targets |
| *Y* | Vector of actual performances |
| *P* | Vector of power-caps of nodes |
| ∆*P* | Vector of changes in power-caps from previous  instance (*k* − 1) to current instance (*k*) |
| *WT* | Matrix of weights on system (*WTsys* ) target and  job (*WTjob* ) targets |
| *W*∆*P* | Matrix of weights on changes in power-caps (∆*P* ) |
| *X*0 | Matrix of current node states at instance *k* |
| *P*0 | Vector of current node power-caps at instance *k* |
| *Q* | Vector of system states for the next *M* intervals |
| *G*, *F* | Control theoretic matrices which map *X*0 and *P*0 to *Y* , respectively |
| *H* , *D* | Control theoretic matrices which map *WT* and  *W*∆*P* to *Q*, respectively |
| *J* | Cost function to be minimized |

Fig. [5](#_bookmark12) provides a visual representation of the state-space model of a node. MATLAB’s system identification tool is used to develop a controllable state-space model, taking into account the uncer- tainties of the system. The figure also shows the equations which govern the state space node model, where *P k* is the node power- cap, *X k* is the state of the node, *Y k* is the IPS of the node, and *D k* is a disturbance signal which accounts for system noise and uncertainties. Matrices A, B, C and V map the corresponding signals to the output signal. These matrices define the system behavior and are identified by the system identification tool based on the mea- sured data. The internal state *X k* of the node gets updated every decision instance based on the active input-output relationship of the currently running job. PERQ uses this model to estimate the performance of a job at maximum power-cap (i.e., TDP) and fair power allocation (*POP* ) in an over-provisioned system.

( )

( )

( ) ( )

( )

* + 1. **PERQ MPC Controller** Next, we proceed to design a con- troller for the above system model. As discussed in Sec. [2.3.1,](#_bookmark7) PERQ employs an MPC controller to optimize the decision cost and attain both targets simultaneously. Expanding on Eq. [1,](#_bookmark9) and specializing it for PERQ, the decision costs related to PERQ’s MPC controller are mathematically represented in Eq. [2,](#_bookmark15) which shows the overall cost of a control strategy, denoted as *J* , during a prediction horizon *M*. The controller predicts the performance of the *NJ* jobs and the system for the next *M* control decision intervals on the *NOP* nodes based on the optimizing strategy. *Tj*,*i Yj*,*i* refers to the difference between the target performance and the actual performance during instance *j* of job *i*, and *WTjob* is the corresponding weight. ∆*Pj*,*i*

−

refers to the change in power-cap on node *i* between instances *j* 1 and *j*, and *W*∆*P* is the corresponding weight. The parameters and variables used in Eq. [2](#_bookmark15) are listed in Table [3.](#_bookmark14)

−

# 3Evaluation

**Methodology:** PERQ evaluation is driven by system parameters and job characteristics of real-world supercomputers. We perform both real-system experiment based evaluation and simulation based

*M*

.

*J* =

*j* =1

.*WTjob*

*NJ*

*j* =1

.

(*Tj*,*i* − *Yj*,*i* )2

+ *W*∆*P*

*NO P*

*i* =1

.

(∆*Pj*,*i* )2

exploration to understand the performance trends of PERQ. Note that performing a real-system experiment on a production super-

computer with power-capping is both time- and cost-prohibitive.

+ *WTsys* (*Tj*,*sys* − *Yj*,*sys* )2 Σ

(2)

To address this, we perform simulation based exploration to gain deeper insights and demonstrate that PERQ performs effectively on a wide variety of real-world system parameters and job characteris-

Representing *J* in matrix form gives the following Eq. [3.](#_bookmark16)

*J* = (*T* − *Y* )*TWT* (*T* − *Y* ) + ∆*PTW*∆*P* ∆*P* (3)

The MPC controller minimizes the cost, *J* , by solving the qua- dratic programming equation shown in Eq. [4](#_bookmark17) (reduced from Eq. [2).](#_bookmark15) It finds the power-caps for all the nodes which minimize the over- all cost for instance *k* + 1. This controller provides a dynamically convergent and stable solution, as was discussed in Sec. [2.3.2.](#_bookmark8)

find *P* to minimize *J* = 1 *PT QP* + *YT P* (4)

2

tics. To drive our simulation study, we use trace of jobs which were executed on Mira and Trinity supercomputers [[2](#_bookmark28), [14](#_bookmark41), [16](#_bookmark43), [27](#_bookmark54)]. In particular, job characteristics such as the number of nodes they are running on and their runtime are used for the simulation. System characteristics such as the number of nodes are also emulated from Mira and Trinity. Mira contains a total of 49,152 IBM PowerPC A2 nodes, while Trinity consists of 19,420 Intel Xeon nodes. For the simulation, we use First-Come-First-Serve (FCFS) with back-filling job scheduling, while making sure that there is always a job avail- able to run at the head of the queue. This is done in order to obtain the true improvement in the job throughput of an over-provisioned system. The statistical distribution of job characteristics such as job size and runtimes are kept same the simulated cluster (Mira

where *Q* = *HTWT H* + *DTW*∆*P D*

and Trinity), but to make the evaluation tractable, the simulation is configured to emulate one day (24 hours) of an actual system.

and *Y* = *HTWT* (*T* − *GX*0 − *FP*0)

Overall, PERQ manages multiple targets by monitoring job char- acteristics and reacting accordingly. PERQ takes the job progress indicators (IPS) as the input, and outputs the power-caps for the nodes, solving the above optimization problem, and sends the new power-cap level to the nodes at every decision instance to reach the optimal and stable solution, as shown in the evaluation (Sec. [3).](#_bookmark18)

* + 1. **PERQ Scope and Limitations** PERQ requires node-level power-capping feature to be enabled in the processor (e.g., Intel’s Running Average Power Limit (RAPL) interface). It may take a few milliseconds to apply and observe the effect of power-capping. Therefore, the control decision interval has to be order of seconds. Our control decision interval is ten seconds, which provides effec- tive performance (Sec. [3).](#_bookmark18) Consequently, PERQ is most effective when jobs run for at least longer than a few minutes to span a few control intervals – as is often case for a productive and useful HPC system; jobs shorter than a few minutes do not require compli- cated power allocation decisions. We note that the PERQ’s decision making is not on the critical path and it makes adjustments at the end of each decision interval for new job arrivals and terminations. The length of PERQ’s decision interval does not need to change dynamically and can be configured as needed. PERQ has no limita- tions or restrictions in terms of a job’s size (i.e., number of nodes it spans). Our evaluation considers real job trace from different super- computers that comprise of different job sizes and runtime lengths. Finally, PERQ’s performance is not dependent on the benchmark suite used for developing the state-space model, but development of this model requires one-time effort for a given node-type. We anticipate this to be a reasonably low investment effort (less than a week of time on only a few nodes – one for each type), for a multi-thousand node cluster expected to run for multiple years. We will make the PERQ controller code publicly available for easier adoption and extension of PERQ inspired approaches.

This translates to 1052 and 1024 jobs on Mira and Trinity systems, respectively, for the over-provisioned system with *f* = 2.0. Note that power consumption profile of the jobs running on these clus- ters is not collected and hence, is not directly available for using as an input to our evaluation. Therefore, the power-performance relationship profiles and phase-behavior of jobs are taken from the application mentioned in Table [1](#_bookmark0) (taken from Exascale Computing Project (ECP) Proxy Application Suite). Each job is assigned the

power-performance characteristics of one of the ten applications using a uniform distribution to have diverse and representative range of behavior. We use the CVXOPT [[3](#_bookmark29)] package to solve the quadratic programming aspect of PERQ’s MPC controller.

The PERQ prototype is deployed on a local HPC cluster: Tardis. A private network is used on Tardis consisting of 16 nodes: one node being the scheduler node (running the python job sched- uling, target setting and controlling features), and others being the cluster nodes (running the actual jobs and performing power- caps). The nodes consist of Intel Xeon E5-2686 processors, and have socket-level RAPL power-capping functionality. The cluster uses Infiniband interconnect for communication over the network. All nodes communicate with the scheduler over a TCP socket about power-cap, IPS, and job start and finish information. Like the simu- lation, the prototype also uses the CVXOPT package to solve the quadratic programming problem every instance, and uses FCFS scheduling. Lastly, a 100 jobs are run for the prototype for each over-provisioning factor and policy, and the benchmarks used for the jobs are taken from Table [1.](#_bookmark0) These runs last for hours on the full cluster and are repeated multiple times to obtain statistically significant results.

We first present simulation based evaluation driven by Mira and Trinity parameters to demonstrate that PERQ works across different job and system characteristics and then, followed by prototype results that support our simulation results and provide research community an implementation prototype for adoption.

20

40% 70%

System Throughput (% Improv. over f = 1)

Mean Performance Degradation (%)

80 15

60

10

40

20 5

0 0

100

80

Max. Performance Degradation (%)

60

40

20

0

FOP

SJS

SRN 196%

PERQ 227%

1.0 1.2 1.4 1.6 1.8 2.0

Over-Provisioning Factor (f)

1.0 1.2 1.4 1.6 1.8 2.0

Over-Provisioning Factor (f)

1.0 1.2 1.4 1.6 1.8 2.0

Over-Provisioning Factor (f)

**Figure 6:** PERQ provides proportionally high system throughput, while being fair for Mira supercomputer parameter-driven evaluation.

100

System Throughput ( )

% Improv. over f = 1

75

50

25

0

40 250

30 200

56% 43%

Mean Performance Degradation (%)

Max. Performance Degradation (%)

150

20

100

10 50

0 0

FOP

SJS 286%

SRN

PERQ

1.0 1.2 1.4 1.6 1.8 2.0

Over-Provisioning Factor (f)

1.0 1.2 1.4 1.6 1.8 2.0

Over-Provisioning Factor (f)

1.0 1.2 1.4 1.6 1.8 2.0

Over-Provisioning Factor (f)

**Figure 7:** PERQ also provides proportionally high system throughput, while being fair for Trinity parameter-driven evaluation.

worst case for a given policy. By definition, FOP enjoys a mean and

**Power Provisioning Policies:** We compare PERQ to several poli- cies. For fairness, we compare it to the *fairness-oriented policy* (FOP), which allocates equal power to all the nodes on the over- provisioned system. For system throughput, we compare it to sev- eral throughput-oriented policies. The first policy prioritizes jobs which are running on the fewest number of nodes: *smallest job size* (SJS). This policy allocates more power to small jobs, anticipating that accelerating them would improve system throughput. Con- versely, one might also imagine prioritizing large jobs (LJS) and helping them finish faster, in order to free up more nodes for other jobs, thus, improving system throughput. However, based on our experiments such a policy actually degrades system throughput. This is because it takes a lot of power to accelerate large jobs, and diverting power to such jobs adversely impacts other concurrently running jobs, slowing them down considerably.

The second throughput-oriented policy prioritizes jobs which are closest to finishing and are running on the fewest number of nodes: *smallest remaining node-hours* (SRN). This policy diverts power to shortest and smallest jobs, knowing that finishing them would improve throughput. It uses *future* knowledge of when the job is going to finish, which is not known for HPC jobs apriori

– users typically overestimate runtime and runtime prediction is prone to inaccuracy. However, we compare PERQ to this policy in order to demonstrate that PERQ provides comparable throughput improvement to a policy which may have prior knowledge and solely focuses on throughput.

**Objective Metrics:** To quantify the system’s performance, we use the system’s *job throughput*, which is the number of jobs which com- plete execution during the duration of the experiment. To assess how fairly the jobs are treated, we use their *mean performance degra- dation*: the mean runtime degradation of *only* the jobs which expe- rience degradation in their runtime with PERQ compared to their runtime with FOP. Note that this metric takes into account only the jobs which experience degradation because considering jobs that benefit from unfairness will skew our assessment of fairness. Jobs which experience better or equal performance as compared to FOP are not considered as they are deemed to be treated fairly. We use the *maximum performance degradation* metric to quantify the

maximum job performance degradation of 0%.

**PERQ improves system throughput significantly, while re- maining fair to jobs.** Fig. [6](#_bookmark19) and Fig. [7](#_bookmark20) show the improvement in system throughput of different power provisioning policies, over worst-case provisioning (*f* = 1) at different over-provisioning factors, for Mira and Trinity supercomputer parameter driven sim- ulations, respectively. First, we observe that FOP and other ad- hoc policies such as SJS are not able to achieve proportional in- crease in system throughput at any over-provisioning factor, i.e., the improvement in system throughput is always less than the

over-provisioning factor. We obtained similar results for LJS (prior- itizing power to largest job size) policy. Second, we observe that a throughput oriented policy such as SRN which accounts for both job size and remaining time can improve throughput, but not pro- portionally. However, *PERQ is able to improve system throughput significantly and proportionally at all over-provisioning factors (f ) and for both Mira and Trinity settings which represent different job characteristics. Notably, PERQ is able to beat the competing policy, SRN, which specifically focuses on improving throughput and has “future" knowledge (job completion time); PERQ does not have access to such knowledge and still provides throughput gains*. Note that

beyond *f* = 2.0, the improvement in system throughput saturates

because the system power budget becomes the bottleneck.

Next, we discuss the fairness performance of PERQ. Fig. [6](#_bookmark19) and Fig. [7](#_bookmark20) also show the mean and maximum performance degrada- tion with different power-provisioning policies at different over- provisioning factors. The throughput-oriented policies have signif- icantly high mean and maximum performance degradation, with SJS performing particularly worse. In fact, SRN, which provides high improvement in throughput, is 2-3x worse in terms of mean and maximum performance degradation compared to PERQ. For ex- ample, for Trinity, the maximum performance degradation of SRN

is over 150% at *f* = 2.0 while PERQ incurs less than 30% maximum performance degradation. Lastly, we note that PERQ achieves up to

50% better system throughput than FOP, while maintaining a mean performance degradation of less than 8% for both Mira and Trinity. This shows that PERQ achieves a large improvement in system

1e9

Power-Cap Target IPS Actual IPS

Power-Cap (kW)

Job IPS

Power-Cap (kW)

4 3

3 2

2 1

0

150

100

50

1e11

1.0 8

Job IPS

Power-Cap (kW)

0.5 6

4

0.0

1e9

6

Job IPS

Power-Cap (kW)

30

4

2 20

0

1e10

2

Job IPS

1

0

0.0 0.2 0.4

Time (hr)

**(a)**

0.0 0.5 1.0 1.5

Time (hr)

**(b)**

0 1 2 3

Time (hr)

**(c)**

0.0 0.5 1.0 1.5

Time (hr)

**(d)**

**Figure 8:** PERQ tracks job-level performance targets effectively for varied jobs, and provides convergent and stable power allocation.

0

System Throughput (% Impov. over Bar 1)

−1

−2

−3

5 10 20 40 60 120

Control Interval (s)

20

15

Mean Performance Degradtion (%)

10

5

0

5 10 20 40 60 120

Control Interval (s)

to distribute it to other jobs, without significantly affecting the performance of this job. We note that eventually, the job is given enough power to meet its target, due to the completion of other jobs and arrival of jobs with different characteristics.

Fig. [8(](#_bookmark21)c) shows a case where a job is given enough power to meet its goal for the most part. However, there are disruptions in between, which can occur when there is a burst of jobs starting

**Figure 9:** PERQ’s throughput is not very sensitive to larger control

intervals.

throughput at a lower over-provisioning factor, while remaining fair compared to ad-hoc and future-knowledge policies.

Interestingly, our results reveal that PERQ is able to achieve similar throughput as FOP with much lower over-provisioning cost. For instance, for Trinity, FOP achieves 80% improvement in system throughput over worst case over-provisioning at *f* = 2.0. However, PERQ achieves the same improvement in throughput at a much lower over-provisioning factor of *f* = 1.4 – thereby, requiring 30% less number of nodes. According to recent data presented by Cray Inc., 47% of the total cost-of-ownership (TCO) of an HPC facility is used as capital expense for purchasing the equipment, while 7% is used as operational expense for the equipment [[38](#_bookmark65)]. Thus, 30% fewer nodes can help reduce the overall cost of equipment by 30%, and the TCO by 16.2%. If these numbers are converted in mone- tary savings, it would result in more than a million dollars saving over the typical life span of a supercomputer [[35](#_bookmark62)] while improving system throughput and maintaining high level of fairness.

**PERQ is convergent, responsive and stable.** Fig. [8](#_bookmark21) shows how a job’s power-cap level and performance under a given target (de- termined based on fairness) evolve over its execution time for four example jobs on Trinity. These examples jobs are selected to cover a wide range in job and PERQ characteristics.

Recall that convergence is the ability of the controller to actually meet the target performance, responsiveness is its ability to reach the target quickly, and stability is the ability to provide constant performance once the target is reached.

Fig. [8(](#_bookmark21)a) shows that PERQ converges to the job-level target quickly (within a few minutes, less than a minute in many cases). In fact, PERQ achieves slightly better performance than the target because it is also trying to meet the system target which requires higher performance from some jobs. We also observe that the job performance is stable once the controller meets the target.

Fig. [8(](#_bookmark21)b) shows a case where the job is not allocated enough power initially to meet its job-level target, but its performance remains close to the target. This is because the controller deems that this power can better serve to meet the system’s or other jobs’ targets. We found that this job has low sensitivity toward power- capping and this allows PERQ to borrow approx. 50kW of power

or jobs finishing. During such periods, the controller responsively tries to figure out the power-caps for the new combination of jobs. In the case of this job, the power-cap is lowered during disruptive phase, but the performance does not get affected considerably.

Lastly, Fig. [8(](#_bookmark21)d) shows a job which is executed during a time period when there is a lot of job start and finish activity, and the job’s power-cap changes frequently during its execution, especially because the controller has observed that this job has low-sensitivity toward power-capping. During such high-activity time periods, the controller readjusts the power-cap gradually and the job’s perfor- mance converges and stabilizes to a new value.

**PERQ performs effectively across different control param- eters and is not sensitive to selection of parameters values.** Our results show that PERQ continues to provide good performance (higher system throughput while meeting fairness goals) as we vary different control parameters (length of decision interval and pre- diction horizon, system throughput improvement ratio, weights on system throughput and ∆*P* ). In fact, our results support that PERQ does not need to highly tune or carefully select these parameters to achieve high performance.

Fig. [9](#_bookmark22) shows the effect of the control interval on the performance of PERQ for Mira supercomputer job trace. Recall that the control interval is the time between two decision instances when the job performances are examined and the nodes are assigned new power- caps. We notice that system throughput degrades minimally even at high control intervals (less than 3%), and mean performance degradation for unfairly treated jobs goes above 5% only at control intervals higher than 40 seconds. We observe similar results for the MPC prediction horizon, which is the number of future control intervals which the controller optimizes its decisions over. So a longer MPC prediction horizon is expected to provide faster con- vergence to the target, but may incur larger computation overhead. We found that that with our chosen control interval length of 10 seconds, PERQ was not sensitive to the length of MPC prediction horizon (i.e., number of control intervals). Note that we avoid con- trol intervals less than 5 seconds to avoid aggressive power-cap switching, even though they are equally effective.

Recall that the system throughput improvement ratio determines how high the system throughput target should be set compared to the over-provisioning factor. The larger this ratio, the higher is the

15

System Throughput (% Impov. over Bar 1)

10

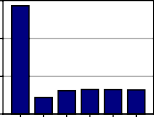
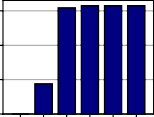
5

0

1 2 4

8 16 32

15

10

Mean Performance Degradtion (%)

5

0

1 2 4 8 16 32

12.5

10.0

System Throughput (% Impov. over Bar 1)

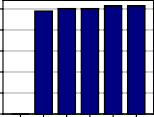
7.5

5.0

0.0

1 2 4 8 16 32

3

2

Mean Performance Degradtion (%)

1

0

1 2 4

8 16 32

0.0

−0.5

System Throughput (% Impov. over Bar 1)

−1.0

−1.5

−2.0



1 5 10 25 50 100

3

2

Mean Performance Degradtion (%)

1

0

1 5 10 25 50 100

System Throughput Improvement Ratio

**(a)**

System Throughput Improvement Ratio

Sys. Throughput Weight

**(b)**

Sys. Throughput Weight

¢*P* Weight

**(c)**

¢*P* Weight

**Figure 10:** PERQ’s effectiveness is not sensitive to control parameters such as (a) system throughput improvement ratio, (b) system throughput weight, and (c) ∆*P* weight on system throughput and job fairness.

100

System Throughput (% Improv. over f = 1)

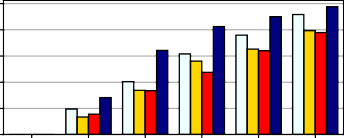
80

60

40

20

0



1.0 1.2 1.4 1.6 1.8 2.0

30

25

Mean Performance Degradation (%)

20

15

10

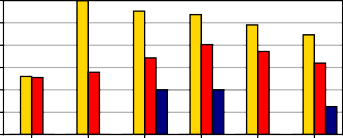
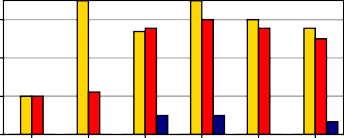
5

0

1.0 1.2 1.4 1.6 1.8 2.0

FOP

60 108%



49%

Max. Performance Degradation (%)

40

20

0

1.0 1.2

SJS

1.4

84%

SRN

1.6 1.8

PERQ

2.0

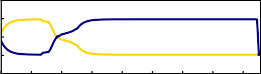
Over-Provisioning Factor (f)

Over-Provisioning Factor (f)

Over-Provisioning Factor (f)

**Figure 11:** PERQ real-system experimental evaluation also confirm that PERQ is more effective than competing policies both in terms of system throughput and fairness.

100



Low Sensitivity App. (ASPA)

High Sensitivity App. (SimpleMOC)

Power-Cap (% of Peak)

75

50

25

0

100

Performance (% of Peak)

75

50

25

0

0 50 100 150 200 250 300 350 400

Time (seconds)

**Figure 12:** PERQ prototype automatically detects applications with different sensitivity toward power-capping and intelligently changes power allocations to meet job-level targets. The second job starts around 50 seconds mark and the first job ends at around 225 seconds mark. Note that the power-cap setting has a minimum limit too (as an idle node still consumes power) and hence, can not be set to zero even after an application finishes its execution.

target for system throughput. If the value of ratio is equal to one, it indicates that the target for system throughput is proportional to increase in the number of over-provisioned nodes in the system. This is conservative and hence, the system throughput improve- ment ratio is set much higher and our results show that PERQ is not sensitive to it once the value chosen is 4 or higher (Fig. [10(](#_bookmark23)a)). Finally, our results also demonstrate that PERQ is not sensitive to selecting the weight placed on meeting the system throughput target or ∆*P* (the amount of change in the power-cap of a node from one control interval to another) (Fig. [10(](#_bookmark23)b) and (c)). Recall that system throughput weight is different than the system throughput improvement ratio. The system throughput improvement ratio de- termines how high the throughput target is set, while the system throughput weight determines how hard the controller should try to meet this target. On the other hand, ∆*P* weight determines how much cost is associated with large ∆*P* , i.e., how must cost is asso- ciated with large switching of the power-caps from one control instance to next. Fig. [10(](#_bookmark23)b) and (c) show that system throughput improvement and fairness metric is not affected significantly even across a wide range of parameter values. Note that PERQ can also

be used to solely focus on improving the system throughput by placing orders of magnitude higher weight on throughput than fairness. We found that following this strategy results in up to 5% system throughput improvement over PERQ, but increases the max- imum performance degradation close to 70%. Finally, we present PERQ prototype results and analysis.

**The PERQ prototype implementation results show that PERQ is an effective strategy in meeting both throughput and fairness targets, and the observed improvements align with simulation results.** Fig. [11](#_bookmark24) shows the system throughput and mean performance degradation of PERQ for different over- provisioning factors. Similar to Mira and Trinity simulation results, PERQ prototype results show that it can deliver up to 25% bet- ter throughput than FOP, while maintaining a mean performance degradation of less than 10%. Other ad-hoc throughput-oriented policies, SJS and SRN, are neither able to improve throughput as much as PERQ, nor are they able to remain as fair as PERQ because of their lack of feedback-based principled approach. In fact, SRN has a mean performance degradation of up 20% (double of PERQ), and maximum performance degradation of up to 60%, while not providing the necessary system throughput improvement.

Lastly, we use the prototype to provide insights into the dynamic behavior of PERQ. Fig. [12](#_bookmark25) shows how the provisioned power trades hands between two different types of applications running on the Tardis HPC cluster. Initially, a low sensitivity application starts run- ning and is allocated as much power as possible, while the empty node is allocated minimum power. At around the 50 seconds time mark, a high sensitivity application starts running on the empty node. The controller determines that it is a high sensitivity applica- tion and gradually transfers power to that application. This does not degrade the performance of the low sensitivity application, while it improves the performance of the high-sensitivity appli- cation. Eventually, the controller completely switches the power allocations of the two applications (at about 150 seconds time mark). The low-sensitivity application continues to perform at close to 100% of its peak performance even at minimum power allocation,

100

MPC Horizon 2 3 4 5

CDF of Total Number of Controller Decisions (%

80

60

40

20

0

0.0

0.1 0.2 0.3 0.4 0.5

Controller Decision Time (seconds)

100

80

MPC Horizon 2 3 4 5

CDF of Total Number of Controller Decisions (%

60

40

20

0

0.0

0.5 1.0 1.5 2.0

Controller Decision Time (seconds)

# 4Related Work

This section provides an overview of prior solutions proposed for efficiently power provisioning on HPC systems. Prior studies have examined the performance of jobs under different power manage- ment techniques [[24](#_bookmark51), [25](#_bookmark52), [61](#_bookmark88)]. However, they do not provide solutions

**Figure 13:** PERQ controller makes its power-capping decision within 0.5 seconds for more than 80% cases: Mira supercomputer (left) and Trinity supercomputer (right).

until it finishes at 55% time mark. This demonstrates how PERQ dynamically detects application characteristics and allocates power in a manner which benefits the system and the jobs.

**Overhead Analysis:** Finally, we discuss the scalability and over- head aspects of PERQ. PERQ has two sources of overheads: (1) MPC controller decision making overheads, and (2) communication of performance indicators (e.g., IPS) from each node to the controller. As we discuss next, our results provide quantitative evidence to show that these overheads are relatively small compared to the control interval. However, more importantly, we first note that these overheads are not on the critical path of application execu- tion. These overheads only delay the optimal power-capping targets for different jobs and hence, these overheads do not pause or in- terrupt the application execution or computing cluster operations. In other words, these overheads simply make the effective length of the control decision intervals slightly longer. Our simulation and experimental evaluation includes these overheads and shows that PERQ is still effective at achieving its goals. Fig. [13](#_bookmark26) shows that MPC controller makes most of its decisions within 0.5 seconds for both the simulated systems Mira and Trinity (with 1052 and 1024 jobs, respectively) for MPC prediction window of length four. As the prediction window size increases, the overhead increases but the overhead still remains fairly low. We also note this overhead is not dependent on the scale of the system, but instead depends on the number of concurrently running jobs and their characteristics. For Trinity, only in less than 2% cases, the MPC controller deci- sion making takes more than one second. Increasing the number of concurrently running jobs in the order of 10,000 can prohibi- tively increase the MPC controller decision making time in some cases due to computationally intensive nature of the calculations. However, several well-known strategies can be applied to alleviate this bottleneck: offloading the computation to GPUs, hierarchical decision making, eliminating the need to perform calculation for every job at every decision instance, creating groups of jobs with similar characteristics. Future efforts can exploit such opportunities to optimize MPC decision controller performance even further.

The second source of overhead (communication of performance metric from the nodes to the controller) was measured to be even lower. We stress tested our system by spawning 100,000 clients in our Tardis cluster and found that communicating such a information from these nodes to controller incurs only 0.19 seconds of delay. Note that we did not perform any optimizations to reduce this delay (e.g., using a dedicated network, performing reductions by filtering redundant information, etc.) that HPC clusters typically perform to monitor system health at much finer granularity and sending it to management workstations. In summary, MPC decision making is the primary source of delay in optimal power-capping control. But, even MPC decision time is relatively small and helps PERQ achieve effective results in terms of fairness and throughput (these delays and corresponding effects are modeled in our simulation results).

to provide performance guarantees under target power budgets.

Many studies have looked into maximizing resource utilization (system job throughput) while satisfying power constraints [[22](#_bookmark49), [46](#_bookmark73), [55](#_bookmark82), [56](#_bookmark83)]. These works are mostly concerned with dynamic appli- cation scheduling to maximize system utilization. Thus, they are compatible with PERQ and can be combined with it. Similarly, job scheduling and placement techniques [[36](#_bookmark63), [57](#_bookmark84)] to maximize perfor- mance under fixed energy budgets are also complementary to PERQ, as PERQ begins to function after the jobs have been scheduled, and is not concerned with scheduling.

Many studies have proposed various uses of dynamic voltage and frequency scaling (DVFS) and power capping to meet power constraints [[7](#_bookmark34), [11](#_bookmark38), [33](#_bookmark60), [52](#_bookmark79)]. However, these solutions (1) do not have a system-level consideration of power budget and therefore, do not take into the account the trade-offs of allocating different amounts of power to different jobs, and (2) they do not provide any feedback- based dynamic functionalities. Some works have also proposed heuristics-based approaches to meet multiple goals [[5](#_bookmark32), [10](#_bookmark37)], while others have proposed monitoring-based solutions to manage the application performance [[23](#_bookmark50), [37](#_bookmark64)]. However, these works have sim- ilar issues as mentioned above related to lack of a global system view to manage multiple jobs and lack of consideration for fairness among the jobs. Several studies have also proposed the use of con- trol theory to manage the power efficiency of applications, while delivering performance guarantees [[13](#_bookmark40), [17](#_bookmark44), [21](#_bookmark48), [37](#_bookmark64), [41](#_bookmark68), [50](#_bookmark77), [61](#_bookmark88), [62](#_bookmark89)]. These studies highlight the effectiveness of control theory in achiev- ing software and hardware targets in HPC applications. However, these studies lack a universal system-level view and continuous estimation of fairness; therefore, are unable to distribute power amongst applications based on what the system can deliver.

# 5Conclusion

This paper described design and evaluation of PERQ, which achieves high system throughput while maintaining job-level fairness. PERQ employs multi-input multi-output control theory which provides theoretic guarantees about satisfying conflicting goals. Extensive real-system and simulation evaluation demonstrate that PERQ im- proves system throughput by up to 50% points, compared to the fairness-oriented allocation policy, while remaining fair to jobs.

**Acknowledgment:** This research is supported by the Northeast- ern University, Amazon AWS research program, and Massachusetts Green High Performance Computing Center (MGHPCC). We are thankful to anonymous reviewers for their feedback. We are espe- cially thankful to Liana Fong for her detailed feedback and efforts which helped us improve the quality of the paper significantly.

# References

1. 2018. Exascale Computing Project Proxy Applications: https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/.
2. George Amvrosiadis, Jun Woo Park, et al. 2018. On the Diversity of Cluster Work- loads and its Impact on Research Results. In *2018 USENIX Annual Technical* *Conference ( USENIX ATC 18)*. 533–546.

{ }{ }

{ }

1. M Andersen, Joachim Dahl, et al. 2013. CVXOPT: A Python Package for Convex Optimization. *abel. ee. ucla. edu/cvxopt* (2013).
2. Reza Azimi, Masoud Badiei, Xin Zhan, Na Li, and Sherief Reda. 2017. Fast Decentralized Power Capping for Server Clusters.. In *HPCA*. 181–192.
3. Prasanna Balaprakash, Ananta Tiwari, et al. 2013. Multi Objective Optimization of HPC Kernels for Performance, Power, and Energy. In *International PMBS* *Workshop*. Springer, 239–260.
4. Alberto Bemporad, Francesco Borrelli, et al. 2000. Optimal Controllers for Hybrid Systems: Stability and Piecewise Linear Explicit Form. In *Decision and Control,* *2000. Proceedings of the 39th IEEE Conference on*, Vol. 2. IEEE, 1810–1815.
5. Sridutt Bhalachandra, Allan Porterfield, et al. 2017. Improving Energy Efficiency in Memory-constrained Applications Using Core-specific Power Control. In *Proceedings of the Workshop on Energy Efficient Supercomputing*. ACM, 6.
6. Abhishek Chandra, Pawan Goyal, and Prashant Shenoy. 2003. Quantifying the benefits of resource multiplexing in on-demand data centers. *Computer Science* *Department Faculty Publication Series* (2003), 20.
7. Hong Chen et al. 1998. A Quasi-Infinite Horizon Nonlinear Model Predictive Control Scheme with Guaranteed Stability. *Automatica* 34, 10 (1998), 1205–1217.
8. Jee Choi, Marat Dukhan, et al. 2014. Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks. In *Parallel and Distributed Processing* *Symposium, 2014 IEEE 28th International*. IEEE, 447–457.
9. Ryan Cochran, Can Hankendi, et al. 2011. Pack & Cap: Adaptive DVFS and Thread Packing under Power Caps. In *Proceedings of the 44th Annual IEEE/ACM* *International Symposium on Microarchitecture*. ACM, 175–185.
10. Howard David, Eugene Gorbatov, et al. 2010. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE International Symposium on* *Low Power Electronics and Design*. ACM, 189–194.
11. Gökalp Demirci, Ivana Marincic, and Henry Hoffmann. 2018. A divide and conquer algorithm for DAG scheduling under power constraints. In *SC18: Inter- national Conference for High Performance Computing, Networking, Storage and* *Analysis*. IEEE, 466–477.
12. Douglas W Doerfler. 2014. *Trinity: Next-Generation Supercomputer for the ASC* *Program.* Technical Report. SNL, Albuquerque, NM (United States).
13. Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power provi- sioning for a warehouse-sized computer. In *ACM SIGARCH computer architecture* *news*, Vol. 35. ACM, 13–23.
14. Dror G Feitelson et al. 2014. Experience with using the parallel workloads archive.

*J. Parallel and Distrib. Comput.* 74, 10 (2014), 2967–2982.

1. Antonio Filieri, Henry Hoffmann, et al. 2015. Automated Multi-Objective Control for Self-Adaptive Software Design. In *Proceedings of the 2015 10th Joint Meeting* *on Foundations of Software Engineering*. ACM, 13–24.
2. Rong Ge, Xizhou Feng, et al. 2005. Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters. In *Supercom-* *puting, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*. IEEE, 34–34.
3. Neha Gholkar, Frank Mueller, et al. 2016. Power Tuning HPC Jobs on Power- Constrained Systems. In *Proceedings of the 2016 International Conference on Par-* *allel Architectures and Compilation*. ACM, 179–191.
4. Neha Gholkar, Frank Mueller, et al. 2018. PShifter: Feedback-Based Dynamic Power Shifting within HPC Jobs for Performance. In *2018 HPDC*.
5. Henry Hoffmann. 2015. JouleGuard: Energy Guarantees for Approximate Appli- cations. In *Symposium on Operating Systems Principles*. ACM, 198–214.
6. Connor Imes et al. 2015. Minimizing Energy under Performance Constraints on Embedded Platforms: Resource Allocation Heuristics for Homogeneous and Single-ISA Heterogeneous Multi-Cores. *ACM SIGBED Review* 11, 4 (2015), 49–54.
7. Connor Imes, Lars Bergstrom, et al. 2016. A Portable Interface for Runtime Energy Monitoring. In *Proceedings of the 2016 24th ACM SIGSOFT International* *Symposium on Foundations of Software Engineering*. ACM, 968–974.
8. Canturk Isci et al. 2006. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Symposium on Microarchitecture*. IEEE, 347–358.
9. Melanie Kambadur and Martha A Kim. 2014. An Experimental Survey of Energy Management Across the Stack. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 329–344.
10. Jaimie Kelley et al. 2016. Adaptive power profiling for many-core HPC architec- tures. In *ICAC 2016*. IEEE, 179–188.
11. Kalyan Kumaran. 2016. Introduction to Mira. In *Code for Q Workshop*.
12. Kien Le, Ricardo Bianchini, Thu D Nguyen, Ozlem Bilgir, and Margaret Martonosi. 2010. Capping the brown energy consumption of internet services at low cost. In *International Conference on Green Computing*. IEEE, 3–14.
13. Jay H Lee. 2009. A Lecture on Model Predictive Control. *Pan American Advanced* *Studies Institute Program on Process Systems Engineering* (2009).
14. Bo Li, Hung-Ching Chang, et al. 2014. The Power-Performance Tradeoffs of the Intel Xeon Phi on HPC Applications. In *Parallel & Distributed Processing* *Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE, 1448–1456.
15. Yang Li et al. 2019. A Scalable Priority-Aware Approach to Managing Data Center Server Power. In *HPCA 2019*. IEEE, 701–714.
16. Harold Lim, Aman Kansal, and Jie Liu. 2011. Power budgeting for virtualized data centers. In *2011 USENIX Annual Technical Conference (ATC)*.
17. Yanpei Liu, Guilherme Cox, et al. 2016. FastCap: An Efficient and Fair Algorithm for Power Capping in Many-Core Systems. In *Performance Analysis of Systems* *and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 57–68.
18. L Ljung. 1999. System Identification-Theory for the User 2nd edition PTR Prentice- Hall. *Upper Saddle River, NJ* (1999).
19. Thomas Ludwig and Manuel Dolz. 2014. Total Cost of Ownership in High Performance Computing. In *Talk at the University of Hamburg*.
20. Olli Mämmelä et al. 2012. Energy-Aware Job Scheduler for High-Performance Computing. *Computer Science-Research and Development* 27, 4 (2012), 265–275.
21. Ivana Marincic et al., Venkatram Vishwanath, and Henry Hoffmann. 2017. PoLiMEr: An Energy Monitoring and Power Limiting Interface for HPC Ap- plications. In *Proceedings of the 5th E2SC Workshop*. ACM, 7.
22. Steven Martin. 2017. Total Cost of Ownership and HPC System Procurement. In

*Talk at the 2017 International Conference in Supercomputing*.

1. David Q Mayne, James B Rawlings, et al. 2000. Constrained Model Predictive Control: Stability and Optimality. *Automatica* 36, 6 (2000), 789–814.
2. David Meisner, Christopher M Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F Wenisch. 2011. Power management of online data-intensive services. In *ACM SIGARCH Computer Architecture News*, Vol. 39. ACM, 319–330.
3. Nikita Mishra, Connor Imes, et al. 2018. CALOREE: Learning Control for Pre- dictable Latency and Low Energy. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating* *Systems*. ACM, 184–198.
4. Frank Mueller, Barry Rountree, et al. 2016. *Power Tuning for HPC Jobs under Manufacturing Variations*. Technical Report. North Carolina State University. Dept. of Computer Science.
5. Bin Nie et al. 2017. Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities. In *MASCOTS* *2017*.
6. Chandrakant D Patel and Amip J Shah. 2005. Cost model for planning, develop- ment and operation of a data center.
7. Tapasya Patki et al. 2013. Exploring Hardware Overprovisioning in Power- Constrained, High Performance Computing. In *Proceedings of the International* *Conference on Supercomputing*. ACM, 173–182.
8. Tapasya Patki et al. 2015. Practical Resource Management in Power-Constrained, High Performance Computing. In *Proceedings of the Symposium on High-* *Performance Parallel and Distributed Computing*. ACM, 121–132.
9. Tapasya Patki et al. 2016. Economic Viability of Hardware Overprovisioning in Power-Constrained High Performance Computing. In *Proceedings of the Workshop* *on Energy Efficient Supercomputing*. IEEE Press, 8–15.
10. Steven Pelley, David Meisner, Pooya Zandevakili, Thomas F Wenisch, and Jack Underwood. 2010. Power routing: dynamic power provisioning in the data center. In *ACM Sigplan Notices*, Vol. 45. ACM, 231–242.
11. Ian R Petersen and Roberto Tempo. 2014. Robust Control of Uncertain Systems: Classical Results and Recent Developments. *Automatica* 50, 5 (2014), 1315–1335.
12. Raghavendra Pradyumna Pothukuchi, Amin Ansari, et al. 2016. Using Mul- tiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual* *International Symposium on*. IEEE, 658–670.
13. Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. 2008. No power struggles: Coordinated multi-level power management for the data center. *ACM SIGOPS Operating Systems Review* 42, 2 (2008), 48–59.
14. Barry Rountree, Dong H Ahn, et al. 2012. Beyond DVFS: A First Look at Per- formance under a Hardware-Enforced Power Bound. In *Parallel and Distributed* *Processing Symposium Workshops & PhD Forum (IPDPSW)*. IEEE, 947–953.
15. Ryuichi Sakamoto, Thang Cao, et al. 2017. Production Hardware Overprovision- ing: Real-World Performance Optimization using an Extensible Power-Aware Resource Management Framework. In *Parallel and Distributed Processing Sympo-* *sium (IPDPS), 2017 IEEE International*. IEEE, 957–966.
16. Ryuichi Sakamoto, Tapasya Patki, et al. 2018. Analyzing Resource Trade-offs in Hardware Overprovisioned Supercomputers. In *2018 IEEE International Parallel* *and Distributed Processing Symposium (IPDPS)*. IEEE, 526–535.
17. Osman Sarood, Akhil Langer, et al. 2014. Maximizing Throughput of Overprovi- sioned HPC Data Centers under a Strict Power Budget. In *Supercomputing (SC)*. IEEE Press, 807–818.
18. Kun Tang et al. 2016. Power-capping aware checkpointing: On the interplay among power-capping, temperature, reliability, performance, and energy. In *DSN* *2016*. IEEE, 311–322.
19. Akshat Verma, Puneet Ahuja, et al. 2008. Power-Aware Dynamic Placement of HPC Applications. In *Proceedings of the 22nd Annual International Conference on* *Supercomputing*. ACM, 175–184.
20. Xiaorui Wang, Ming Chen, Charles Lefurgy, and Tom W Keller. 2012. Ship: A scalable hierarchical power control architecture for large-scale data centers. *IEEE* *Transactions on Parallel and Distributed Systems* 23, 1 (2012), 168–176.
21. Will Whiteside, Shelby Funk, Aniruddha Marathe, and Barry Rountree. 2017. PANN: Power Allocation via Neural Networks Dynamic Bounded-Power Allo- cation in High Performance Computing. In *Proceedings of the 5th International* *Workshop on Energy Efficient Supercomputing*. ACM, 8.
22. Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, San- jeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. 2016. Dynamo: facebook’s data center-wide power management system. In *2016 ACM/IEEE 43rd Annual* *International Symposium on Computer Architecture (ISCA)*. IEEE, 469–480.
23. Huazhe Zhang and Henry Hoffmann. 2016. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *ACM* *SIGARCH Computer Architecture News* 44, 2 (2016), 545–559.
24. Huazhe Zhang and Henry Hoffmann. 2018. Performance & energy tradeoffs for dependent distributed applications under system-wide power caps. In *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 67.