Spring 5-20-2018

Towards Mitigating Co-incident Peak Power Consumption and Managing Energy Utilization in Heterogeneous Clusters

Renan DelValle Rueda
State University of New York at Binghamton, rdelval1@binghamton.edu

TOWARDS MITIGATING CO-INCIDENT PEAK POWER CONSUMPTION AND MANAGING ENERGY UTILIZATION IN HETEROGENEOUS CLUSTERS

BY

RENAN DELVALLE RUEDA

BS, Binghamton University, 2012
MS, Binghamton University, 2014

DISSERTATION

Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer Science
in the Graduate School of
Binghamton University
State University of New York
2018

Accepted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer Science
in the Graduate School of
Binghamton University
State University of New York
2018

August 10, 2018

Madhusudhan Govindaraju, Chair and Faculty Advisor
Department of Computer Science, State University of New York at Binghamton

Michael J. Lewis, Member
Department of Computer Science, State University of New York at Binghamton

David Liu, Member
Department of Computer Science, State University of New York at Binghamton

Alexey Kolmogorov, Outside Examiner
Department of Physics, State University of New York at Binghamton

Abstract

The infrastructure for processing and analyzing high volumes of data has evolved at a fast pace to keep up with the large collection of user generated data. As a result, new technologies have emerged to facilitate the processing of data as well the management of the datacenters used by these data processing tools. As a cluster manager, Apache Mesos falls into the latter category. Mesos works by creating an abstraction layer that allows multiple frameworks to share resources from a vast collection of nodes. In contrast to models in which resources were statically reserved for each framework, Mesos acts as a broker for traditional resources: CPU, Memory, and Disk Space. With this model, each framework is capable of dynamically using as many or as few resources as needed during execution. The Mesos model was created with the goal of increasing hardware resource utilization in datacenters. However, the model does not account for the increase in energy and power consumption that are created as a direct consequence of high utilization. Peak power is important to consider due to the cost of cooling [17], the physical limitations of power delivery [59], as well increasing costs per Watt of peak critical power factored in when building a datacenter [69]. Furthermore, there are scenarios where reducing power demand for a specific time period can result in costs savings. For example, reducing power demand during hours designated by power providers as peak hours, where the cost per Watt is several

times the base rate [42], can reduce operational costs. Thus, as the utilization of the available resources continues to increase due to the Mesos resource allocation model, peak power consumption and energy consumption quickly become important factors to consider.

Our first contribution [16] presents a proof of concept by showing there exist a difference in peak power consumption and energy utilization when changing the order in which workloads of varying power consumption are scheduled. In this work we use profiles created from offline executions of a series of benchmarks to determine the order in which each workload is executed on the cluster. One strategy seeks to schedule workloads in non-decreasing order of peak power consumption while the other strategy gives priority to workloads that use more power on average than the median average power consumption of all benchmarks. For both of these policies, each node is treated as a bin where the sum of the chosen power based metric for all workloads selected to run on a node is less than or equal to the Thermal Design Power of that node. Both policies are applied within a local scope (a single node) and within a global scope (whole cluster). Furthermore, we demonstrate how a simple static power cap can help to reduce both power and energy consumption for these workloads.

Our second contribution [14] expands upon our previous work by introducing a power aware Mesos framework, Electron. Electron contains implementations of various scheduling algorithms as well as a dynamic power capping policy. Using an expanded set of workloads, including cryptographic and scientific applications, we quantified the effects that different power capping strategies have on different scheduling policies. Aurora's first fit policy was used as a point of comparison for the performance of each combination. Further more, we introduced *extrema*, a dynamic power capping policy capable of reacting to cluster power consumption in order to reduce peak power and energy consumption.

Finally, our third contribution [15] builds upon our previous work by introducing two new scheduling policies and a modified version of extrema, *progressive extrema*. Progressive extrema is introduced to address extrema's inability to continue to reduce power and energy consumption once every node in the cluster has been power capped. The additional scheduling policies offer further insight into how scheduling policies affect makespan, energy consumption, and power consumption. Workloads run under each different scheduling algorithm and power capping policy combination were also varied to quantify how the makeup of a workload affects the overall power and energy consumption when being performed under different scheduling and capping combinations.

Dedicated to my mom Carmen Araya, my late grandparents Waldo Rueda Jara and Carmen Saez Sepúlveda, and my late step father Luis Araya.

Acknowledgments

In March 5th, 2000, the day after my tenth birthday, my mom and I landed in New York City. We arrived at the airport with two suitcases, a few personal belongings, and a laundry list of hopes and dreams. Looking back on that moment, writing this acknowledgement section to thank all the people that helped me along the way to getting a PhD feels surreal.

There are many folks without whom this thesis would not be possible. Their help, guidance, support, and patience have proved crucial to this achievement.

From my days living in Port Washington, NY: Tony Boccia, Steve Friedman, and Warren Herz, my football (soccer) coaches. Thank you for going out of your way to provide me with a platform to integrate into the US while playing a sport I continue to love. Joan Budny and her family for always making me feel like I was just another member of the clan. The Sweeney family for their support. Bob Joseph, my 6th grade teacher, and the first teacher to tell me I could achieve great things and get me to believe in myself. Jim Jones, my 9th Biology teacher, for fostering my love for science. My AP European History teacher, Petro Macrigiane, for believing in me and always pushing me to be a to achieve more academically. My strength training track and field coach, Doug Matina. Doug taught me the importance of developing my mental strength and to never give up, no matter who or what is ahead. My track and cross country coach, AP US History teacher, fellow Binghamton alumni, and mentor, Jeremy Klaff, for being a great teacher, both in the class room and in life. Thanks for your life lessons, for looking out for me, and for helping me be a better person. My former cross country and track and field teammates. Completing a PhD felt a lot like running an endless race. I often thought back to the days we all spent running together and remembered how good it felt to just finish the race, especially when you felt the odds were stacked up against you. You guys continue to be a daily source of inspiration as I see you all achieve great milestones in your careers and personal lives.

From my days at Binghamton University: My mentor, Jessica Hartog, who played a key role in helping me choose an area of study, mentored me, and taught me how to write technical papers. My labmates Abhishek Jain and Pradyumna Kaushik, who selflessly donated hours upon hours of their free time as Master's students to my research. I wouldn't be writing this without your constant support and friendship. Professors Leslie Lander, Eileen Head, Patrick Madden, David Liu, Mike Lewis, and Tim Miller for their friendship, dedication, teachings, and guidance. Craig Stewart and all the folks involved with Jetstream at Indiana University who gave me the opportunity to play a small part in helping launch one of the first academic clouds.

From my friends and family: My mom Carmen for her support, her unconditional love, and her many sacrifices that made this a reality. Thank you for leaving the comfort of all that you knew in search of better opportunities for me. My cousin Felipe, his partner in crime Maria Fernanda, my two nieces, Sofi and Leo, my aunt Michelina, and my cousin Jose Eduardo for their constant love and support from more than eight thousand kilometers away in Chile. My step father, Luis, who is no longer with us but I know would be extremely proud of this accomplishment. The

Ritchie and Wasserman family for fostering my love of computers by letting me build my very first computer for them. My friends, old and new, who have helped me along the way. You guys are a big part of why I was able to write this.

My current and former colleagues at PayPal: My manager Meghdoot for allowing me to come back as an intern three (yes, three) times while I was a PhD candidate. My co-worker Misho for pushing me to finish this degree when motivation was at an all time low. The rest of the Genesis team for the support, advice, and encouragement along the way. Without my internships at eBay/PayPal this thesis would not exist.

The folks at the Apache Aurora project, without whom my work presented here would not have been possible.

Last but not least my advisor, Madhusudhan Govindaraju, who always found a way to keep the process engaging and entertaining.

I'd also like to acknowledge the National Science Foundation (NSF) funded Louis Stokes Alliances for Minority Participation (LSAMP) Bridges to Doctorate (BD) program as well as the Clifford D. Clark Diversity Fellowship for providing me with funding which has allowed me to achieve this goal.

Table Of Contents

List of Tables **xiii**

List of Figures **xiv**

List of Abbreviations **xxii**

1 Introduction **1**

1.1 Apache Mesos 2

1.2 Apache Aurora 3

1.3 Contributions 5

1.4 Thesis Statement 6

2 Related Work **7**

3 Exploring the Design Space for Optimizations with Apache Aurora and Mesos **13**

3.1 Experimental Setup 14

3.2 Max Peak First Bin Packing (MPF-BP) 17

3.3 Max Average Peak Bin Packing (MAP-BP) 22

3.4 MPF-BP for the Entire Cluster 26

3.5 MAP-BP for the Entire Cluster 28

3.6 Analysis of our Policies on Aurora and Mesos 30

3.7 Conclusion 31

4 Electron: Towards Efficient Co-Scheduling on Heterogeneous Clusters with Apache Mesos 33

4.1 A new Framework 35

4.2 Consuming Mesos Offers 37

4.2.1 First Fit 37

4.2.2 Bin Packing 37

4.3 Measuring and Limiting Power Consumption 38

4.3.1 Static Power Capping 38

4.3.2 Extrema Dynamic Power Capping 39

4.4 Experimental setup 40

4.5 Performance Analysis 43

4.5.1 Effects of Static Power Capping 47

4.5.2 Effects of Extrema Power Capping 49

4.6 Conclusion 54

5 Exploiting Efficiency Opportunities Based on Workloads with Electron on Heterogeneous Clusters 58

5.1 Introduction 59

5.2 Electron 61

5.2.1 Power Classes 63

5.2.2 Consuming Mesos Offers 63

5.2.3 Power Capping Policies 66
5.3 Experiments 70
5.3.1 Setup 70
5.3.2 Workloads 72
5.3.3 Median of Medians Max Power Usage 72
5.3.4 Task Classification 75
5.4 Performance Analysis 75
5.4.1 Performance of Scheduling Policies 76
5.4.2 Power Capping Impact On Scheduling 82
5.5 Conclusion 92

6 Conclusions and Future Work 95
6.1 Conclusions 95
6.2 Future Work 96
6.2.1 Watts As A Resource (WaaR) 96
6.2.2 Adaptive Power Aware Scheduler 101
6.2.3 CPI based Power Aware Scheduler 103
6.2.4 Fine-grained Extrema 103
6.2.5 Considering Streaming Workloads 104

Bibliography 105

List of Tables

3.1 List of benchmarks in the DaCapo Suite 16

4.1 Power consumption statistics for Aurora, Electron FF, and Electron BP. 43

4.2 Power consumption statistics for Aurora, Electron FF, and Electron BP under a static power cap. Electron BP experiences the highest power peaks and makespan but the lowest mean, median, 90^{th} percentile, and 95^{th} percentile. 47

4.3 Power consumption statistics for Electron FF and Electron BP under an extrema dynamic power cap. The only metric in which Electron FF bests the performance of Electron BP is on makspan. 50

5.1 Workload benchmarks 73

5.2 Comparison of the effects of different scheduling policies for a Light Power Consuming Workload. 77

5.3 Comparison of the effects of different scheduling policies for a Moderate Power Consuming Workload. 78

5.4 Comparison of the effects of different scheduling policies for a Heavy Power Consuming Workload. 80

List of Figures

1.1 Architecture of Mesos and Aurora using the Thermos Executor. . . . 2

3.1 Baseline node power consumption. Outlines the effects of power capping the baseline node. The max peak power is reduced using a power-cap of 50% of the total TDP of the node 19

3.2 Baseline node Memory utilization. This figure shows that compared to the default run, the MPF-BP optimized and uncapped run achieved a 48% increase in total memory utilization, while the power-capped-bin-packed achieved a 31% total memory use increase compared to the default run. 20

3.3 Baseline node Energy consumption. 21

3.4 Baseline node Power consumption using MAP. The bin-packed only run shows similar power peaks to those observed in Figure 3.1. The most noticeable difference is the decrease in consumption around 100 seconds and a late spike at around 150 seconds. The power-capped-bin-packed run displays similar characteristic with lower peaks by 90% when compared to the baseline and 1% compared to the optimized run. 23

3.5 Baseline node Memory Utilization using MAP. Memory utilization, shown in Figure 3.5, shows that compared to the default run, the MPF-BP optimized and uncapped run achieved a 85% increase in total memory utilization, while the power-capped-bin-packed achieved a 67% total memory use increase compared to the default run. 24

3.6 Baseline node Energy consumption using MAP. MAP-BP, Baseline Node using the MAP-BP policy. The energy utilization compared to the MPF-BP is 14% higher for the bin-packed-only run and 17% higher for the power-capped-bin-packed run. 25

3.7 Power Utilization of cluster using MPF-BP. The power usage for the entire cluster shows a similar trend to the one seen in Figure 3.1 The max power peak for the default run, however, is 3% smaller than the max peak of the cluster running the MPF-BP. The max peak for the power-capped-bin-packed run shows an improvement over the other two runs by 8% against the default run and 11% against the power-capped-bin-packed run. 26

3.8 Memory Utilization of cluster using MPF-BP.Memory utilization is, at its highest, 147% higher than the bin-packed only run and 157% higher than the power-capped-bin-packed run. The default run expends 77% more energy than the bin-packed only run and 86% more energy than the power-capped-bin-packed run. 27

3.9 Energy Consumption of cluster using MPF-BP. 28

3.10 Power consumption of cluster using MAP-PB.The power trend for the bin-packed only and power-capped-bin-packed follow similar patterns, with the difference in peaks being 11%. The peak power for the bin-packed only improves upon the bin-packed only max peak in Figure 3.7 by 10% while the power-capped-bin-packed improves upon the power-capped-bin-packed max peak in Figure 3.7 by 10%. 29

3.11 Energy consumption of cluster using MAP-BP. The energy utilization is similar to the MPF-BP policy. 30

4.1 Architecture of Mesos and Aurora/Electron. Note that Electron has the same position in the architecture as Aurora. 36

4.2 Cluster-wide CPU & Memory Utilization of Aurora and Electron without a power cap. 44

4.3 Cluster-wide Power Consumption of Aurora and Electron without a power cap. 45

4.4 Cluster-wide Energy Consumption of Aurora and Electron without a power cap. 46

4.5 Cluster-wide CPU & Memory Utilization for Aurora and Aurora running under a static power cap. 48

4.6 Cluster-wide Power Consumption for Aurora and Aurora running under a static power cap. 49

4.7 Cluster-wide Energy Consumption for Aurora and Aurora running under a static power cap. 50

4.8 Cluster-wide CPU & Memory Utilization for Electron First Fit, Electron First Fit running under static cap, and Electron First Fit running under the extrema dynamic cap. 51

4.9 Cluster-wide Power Consumption for Electron First Fit, Electron First Fit running under static cap, and Electron First Fit running under the extrema dynamic cap. 52

4.10 Cluster-wide Energy Consumption for Electron First Fit, Electron First Fit running under static cap, and Electron First Fit running under the extrema dynamic cap. 53

4.11 Cluster-wide CPU & Memory Utilization of uncapped Electron Bin Packed, statically capped Electron Bin Packed and Electron Bin Packed running under Extrema capping. 54

4.12 Cluster-wide Power Consumption of uncapped Electron Bin Packed, statically capped Electron Bin Packed and Electron Bin Packed running under Extrema capping. 55

4.13 Cluster-wide Energy Consumption of uncapped Electron Bin Packed, statically capped Electron Bin Packed and Electron Bin Packed running under Extrema capping. 56

5.1 Detailed architecture of Electron and Mesos. 62

5.2 Distribution of tasks by classification for a Moderate Power Consuming workload when using Bin-Packing. The number of nodes for each power class is shown in parentheses. This figure shows Class A nodes processing more power intensive tasks than Class D nodes despite having half as many workers when using the Bin-Packing strategy. 67

5.3 Distribution of tasks by classification for a Moderate Power Consuming workload when using Max-Min. The number of nodes for each power class is shown in parentheses. This figure shows Class D bearing a larger burden than the rest of the classes with Max-Min as the scheduling strategy. 68
5.4 Distribution of tasks by classification for a Moderate Power Consuming workload when using Max-GreedyMins. The number of nodes for each power class is shown in parentheses. This figure shows Class A and Class D completing about the same number of tasks in number with Class D processing more power intensive tasks when using a Max-GreedyMins strategy to schedule workloads. 69
5.5 Power consumption of Light Power Consuming Workload when scheduled with different scheduling policies. 76
5.6 Power consumption of Moderate Power Consuming Workload when scheduled with different scheduling policies. 79
5.7 Power consumption of Heavy Power Consuming Workload when scheduled with different scheduling policies. 81
5.8 Uncapped workloads. Comparison of Cluster-wide Energy Consumption of different scheduling policies when used to schedule different classes of workloads (Light, Moderate and Heavy). 83
5.9 Extrema capped workloads. Comparison of Cluster-wide Energy Consumption for different scheduling policies and classes of workloads in combination with the Extrema dynamic power capping strategy. . . . 84

5.10 Progressive Extrema capped workloads. Comparison of Cluster-wide Energy Consumption for different scheduling policies and classes of workloads in combination with the Progressive Extrema dynamic power capping strategy. 85
5.11 Power profile of Light Workload run under Extrema capping. 86
5.12 Power profile of Moderate Workload run under Extrema capping. . . 87
5.13 Power profile of Heavy Workload run under Extrema capping. 88
5.14 Power profile of Light Workload run under Progressive Extrema capping. 89
5.15 Power profile of Moderate Workload run under Progressive Extrema capping. 90
5.16 Power profile of Heavy Workload run under Progressive Extrema capping. 91

6.1 Cluster-wide CPU Utilization when WaaR is deployed and the cryptography benchmark is included. Electron FF-WaaR and Electron BP-WaaR use far less power and CPU time, but suffer a larger runtime. . 97
6.2 Cluster-wide Power Utilization when WaaR is deployed and the cryptography benchmark is included. The Power ceiling is lowered by the deployment of Watts as a Resource. Cryptography is the highest Power consuming benchmark in our set, causing it to starve when we decrease the number of Watts as a resource available in the cluster. 98
6.3 Cluster-wide Memory Utilization when WaaR is deployed and the cryptography benchmark is included. 99
6.4 Cluster-wide CPU Utilization when WaaR is deployed but the cryptography benchmark is not included. 100

6.5 Cluster-wide Power Utilization when WaaR is deployed but the cryptography benchmark is not included. 101
6.6 Cluster-wide Memory Utilization when WaaR is deployed but the cryptography benchmark is not included. 102

List of Abbreviations

AHP	Average Historical Power
BP	Bin Packing
CPI	Cycles Per Instruction
CPU	Central Processing Unit
FF	First Fit
HPC	High Performance Computing
HPCW	Heavy Power Consuming Workload
JVM	Java Virtual Machine
LPCW	Light Power Consuming Workload
MAP-BP	Max Average Peak Bin Packing
MGM	Max Greedy Mins
MM	Max Min
MPCW	Moderate Power Consuming Workload

MPF-BP	Max Peak First Bin Packing
NFS	Network File System
PCP	Performance Co-Pilot
RAM	Random Access Memory
RAPL	Running Average Power Limit
TDP	Thermal Design Power
VM	Virtual Machine

Chapter 1

Introduction

Datacenters (DC) are a tremendous source of power consumption throughout the world. In 2014, the Natural Resources Defense Council (NRDC) released a report that estimates DC energy consumption for 2013 to be around 91 billion kilowatt-hours. Furthermore, the report predicts that energy consumption in the US alone reach 140 billion kilowatt-hours annually by 2020. [11]. Power consumption by servers in a datacenter is further compounded by the costs of cooling [50] and power delivery [52]. As microservices and large scale data analysis continues to move into the cloud, datacenters powering these services should strive to operate in the most efficient manner possible. This means achieving high utilization of hardware resources while operating in a manner that is both energy conscious and power efficient.

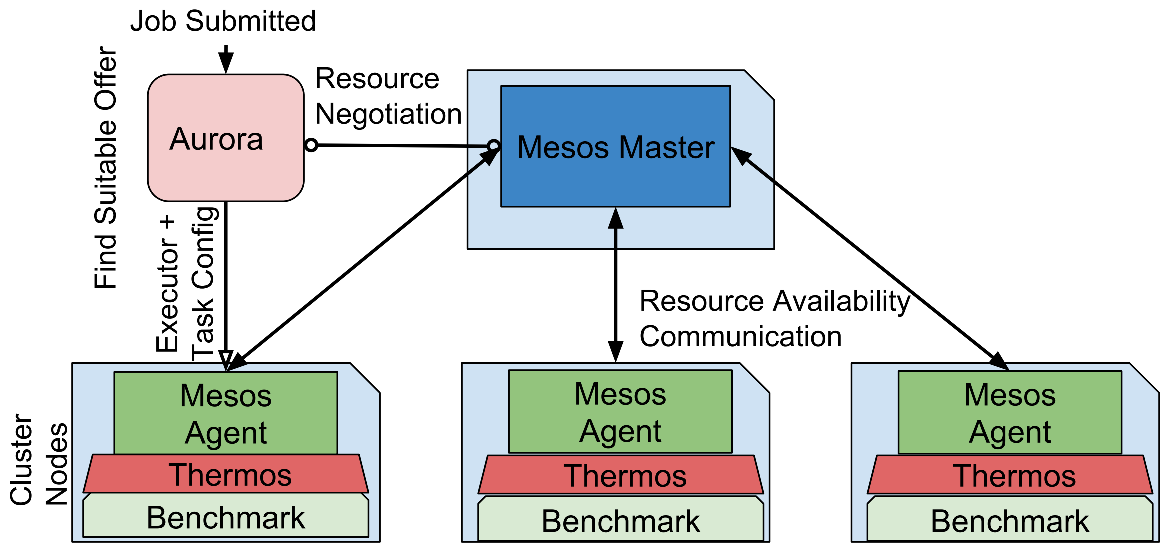

Figure 1.1: Architecture of Mesos and Aurora using the Thermos Executor.

1.1 Apache Mesos

Mesos provides scalability to massive scale applications. Examples of its use include Apple's Siri [72], Bloomberg's data analytics [23], Paypal's continuous integration system [65], Netflix's container orchestration [39], and cloud infrastructure management at Verizon Labs [64]. IBM has explored integrating enterprise policies by way of IBM EGO (enterprise grid orchestrator) into Mesos [2] for potential future use. Apache Mesos provides a layer of abstraction above the compute resources in data centers and large clusters. Mesos allows *frameworks* to deploy tasks onto nodes. An *executor* responds to a Mesos defined API to establish and maintain registration with Mesos agents, launch tasks onto resources that are in control of the executor's framework, clean up after tasks that fail, and terminate all tasks on a resource that is being rescinded.

With the default installation, Mesos ecosystem works as follows:

- A *worker* daemon analyzes the machine on which it runs, determines available resources and advertises them to the Mesos master.

- The Mesos *master* partitions the pool of resources and makes them available to registered frameworks by sending *Resource Offers*. A framework may refuse an offer if it does not suit its needs or consume it by launching tasks on the node linked to the offer.

- Mesos allocation strategies are used to determine how resource offers are made to frameworks. By default, Mesos uses the *Dominant Resource Fairness (DRF)* allocator which encourages fairness between frameworks by striving to allocate sufficient resources to each framework[22].

Furthermore, the Mesos abstraction layer can be configured to be fault tolerant by deploying multiple master instances. The system will elect a master to be a leader through Zookeeper, a distributed key-value store [27] that utilizes a Paxos [37]-like algorithm called Zab (ZooKeeper Atomic Broadcast) to maintain consistency. [32] If the leading master experiences a failure, a new leader will be chosen from the remaining master nodes.

1.2 Apache Aurora

Apache Aurora [4] is a framework that runs on Apache Mesos as shown in Figure 1.1. It allows an application to be scheduled on nodes belonging to a Mesos cluster. Just like Mesos, Aurora is able to be configured to run in a fault tolerant mode by running multiple instances of the scheduler and electing one instance as the master scheduler through Zookeeper [27].

Aurora has its own Domain Specific Language (DSL) that allows users to configure a Job. A Job consist of a collection of Tasks, each of which is comprised of Processes. Each process is understood and managed by the Thermos executor bundled with Apache Aurora. Thermos is run as a Mesos task on worker nodes and is responsible for launching and monitoring each process.

Job Life-Cycle with Aurora: A job configuration is first submitted to Aurora. The configuration contains resource requirements, number of instances to run, and may contain constraints (e.g. task(s) can only be run on a specific node). Depending on the configuration, one or multiple tasks begin their life-cycle in the *Pending* state. In this state the scheduler seeks to find a resource offer made by Mesos that appropriately matches the resource requirements and constraints defined by the job configuration for each task. When an adequate resource offer is found, the task moves into the *Assigned* state. A Remote Procedure Call (RPC) is made to a Mesos Agent containing the task's configuration. A Thermos executor is launched and an acknowledgment is sent back to the Aurora scheduler. Upon receiving the acknowledgment the scheduler moves the task into the *Starting* state. During the *Starting* state a sandbox is created in the worker node chosen to execute the task. Upon the successful creation of a sandbox the task enters the *Running* state where it remains until it enters the *Finished* state if it is able to run to completion or the *Failed* state if not.

Jobs in Aurora can be of type Service, Ad-hoc, or Cron. Services are always restarted after entering into a terminal state. Ad-hoc jobs are only restarted if it enters a Failed state. Cron jobs are run at specific intervals of time. For this body of work only ad-hoc jobs are considered as Services are not meant to finish executing which makes it difficult to measure the overall energy used.

1.3 Contributions

- We demonstrate how bin-packing a set of tasks can effectively reduce the peak-power usage and total energy usage while increasing the node utilization when workloads are co-scheduled to be run using Mesos and Aurora.
- We present Electron, a Mesos framework that shows how a combination of scheduling policies and power capping strategies leads to lower peak power draws and/or lower energy consumption.
- We quantify the effect of Electron on resource utilization (CPU and memory) and makespan. We show how an increase in resource utilization does not have to result in an increase of power and/or energy consumption.
- We demonstrate the strengths and weaknesses of two fundamental strategies for selecting Mesos offers. A First Fit strategy provides computationally fast, short-sighted scheduling, while Bin Packing based strategies are computationally complex but are capable of providing power efficient task distribution. While we demonstrate the use of select policies to show the efficacy and viability of Electron, several other policies proposed in the literature can also be applied for use with Mesos via our framework.
- We quantify the impact of using RAPL to cap a cluster while using First Fit or Bin Packing based Mesos offer selecting strategies. We show how applying a blanket static power capping policy results in increased makespan when used in combination with Bin Packing task distribution strategies.

- We present the *Extrema* and *Progressive Extrema* dynamic power capping strategies which are able to reduce peak power consumption in a cluster while having little to no impact on makespan. Extrema is able to reduce coincidence peak power while having a subdued impact on makespan while Progressive Extrema focuses on achieving lower power peaks than Extrema.

- We present *Max-GreedyMins* a high throughput Bin Packing scheduling strategy that is best reserved for a handful of high power intensive tasks scheduled on the same cluster as a plethora of lower power intensive tasks.

- We study three workloads of varying power intensiveness as well as the effect of scheduling policy and power capping strategy on power and energy consumption for these workloads. We demonstrate how Extrema is best suited for light and moderate power consuming workloads while Progressive Extrema is better suited for heavy power consuming workloads due to its aggressive power capping approach.

1.4 Thesis Statement

This thesis presents a study on techniques that allow for the mitigation of co-incident peak power draws as well as managing energy consumption in heterogeneous clusters. By deploying dynamic power capping policies on top of different scheduling algorithms we are able to achieve lower power peaks and, in many cases, lower energy consumption. Furthermore, this work shows how the efficiency of the power capping and scheduling policy combination on lowering peak power consumption greatly depends on the types of workloads being executed.

Chapter 2

Related Work

Several efforts have been directed towards dividing a cluster into hot or cold zones or determining a Covering Subset such that the cluster contains at least one replica of each data block and the rest of the cluster can be shut down or put in sleep mode [34, 40, 38]. These schemes allow some nodes to be shut down or suspended. Lang and Patel [38] built on this idea such that instead of leaving the cluster online at all times with some nodes in sleep mode, they evaluate the savings of booting up nodes once a job arrives. These approaches are not practical in Data Centers, as they negatively affect the responsiveness of the system, especially for unexpected workloads [47].

Abdelzaher et al. [1] use CPU temperature for scheduling jobs on a MapReduce cluster. In our earlier work on MapReduce based frameworks [24], we also quantified the relationship between CPU temperature and energy consumption and adapted the MARLA MapReduce framework [18] to dynamically schedule work to the nodes in a heterogeneous cluster. While these results work for CPU intensive workloads, they do not directly apply to Mesos based clusters that also take other parameters such as storage and memory use into account for scheduling.

Karakoyunlu et al. [33] developed a 2-approximation solution for minimizing energy consumption and balancing system load. Their schemes (Fixed Scheme, Dynamic Greedy Scheme, and Correlation-Based Scheme) are designed to link cloud storage needs to the cloud users. These policies can be adapted depending on the priorities set by the cloud storage system. Our work takes a similar approach but our policies are applied to affect the scheduling on Mesos clusters using CPU, memory, and power usage.

Bodas et al. [7] developed a power aware scheduler for SLURM [30]. The power consumption of each node is monitored and a uniform frequency mechanism is used to limit power. Similar to our work, their design allows a policy-driven approach for high and low power consumers. This work relies on changing the processor frequency across nodes on which benchmarks are being run. Our approach, on the other hand, assumes that in large data centers, where jobs are co-scheduled on virtualized clusters, modification of the default CPU frequency scaling mechanisms is not practical.

Yang et al. [74] propose a policy driven, knapsack based scheduling algorithm for a power aware scheduler with the goal of reducing the electricity bill without degrading the system utilization. Their scheduling algorithm dispatches jobs with higher power consumption during the off-peak period and jobs with lower power consumption during the on-peak period. While this approach works for job batches that do not require high throughput. Our work focuses on is on scheduling jobs that are currently in queue to be launched on Mesos as quickly as possible. Our framework can be extended to incorporate their policies for batch jobs.

Sarood et al. [63] propose a software-based online resource management system and a scheme that uses an adaptive runtime system that can dynamically change the resource configuration of a running job. As a result resources allocated to currently executing jobs can be modified. While this work also allows power capping, it assumes job malleability such that resources can be dynamically changed for a running job. This greatly increase the complexity of managing resources in a cluster and poses scalability challenges. In our work, we assume that the resources allocated to a job cannot be changed as that is a tenet of Mesos's design.

Li et al. [41] designed an energy aware task scheduling algorithm base on the Energy Aware Min-Min algorithm (EAMM) [43] where they put forward an Online Power-saving State Control Strategy (OPSCS). This strategy assumes the nodes in the cluster to be in one among k states, the k^{th} state being the most idle state using the least amount of power and the 0^{th} state being the most active state using the most amount of power. According to the EAMM algorithm, a node switches from a lesser active state to the most active state when a task is set to run on that node. The state of a node is stepped down one state at a time. Each step down transition is a result of a node being idle in its current state for a longer time than a defined threshold. The EAMM algorithm is not well suited for handling large amounts of workload as the probability of a node stepping down to a lesser active state reduces as the amount of workload increases. Furthermore, the work assume that for each task being scheduled, the target node has transition to the most active state. This strategy may not yield an optimal result for a less power intensive tasks. The EAMM algorithm was implemented using a simulator assuming that the CPU states can be controlled. In contrast, our work assumes that in large data centers that use Mesos, it is not practical to expect system administrators to allow CPU state or frequency modifications.

Galloway et al. [21] present a power aware load balancing algorithm that maintains the states of all the compute nodes and the resource utilization percentages, using these metrics to make a decision on the number of operating compute nodes accordingly. Their algorithm places a job on a compute node that is powered on and can accommodate the job based on the size of the requested resources. Otherwise, a shutdown compute node is powered back up to accommodate this new job. Their algorithm shuts down a compute node when its resource utilization is less than 25%. This approach is not practical in Data Centers as the system's responsiveness takes a hit on the arrival of an unexpected job and job migration might have an unexpected effect on power consumption.

Wu et al. [73] take aim at a different problem regarding power consumption: the underutilization of power. Using Dynamo, a data center-wide power management system that monitors the entire power hierarchy and makes coordinated control decisions to safely and efficiently use provisioned data center power, they seek to use power available to the datacenter efficiently. Dynamo works in a similar fashion to our Extrema dynamic power capping algorithms. Where it differs from our work is in their use of on-board power sensors and meters. Our work operates on the assumption that access to the physical datacenters is not possible, thus placing power meters where they aren't already deployed becomes challenging. In order to overcome this hurdle, we use the power estimations provided by RAPL to allow our power capping algorithms to make decisions. Furthermore, our work deploys the power capping algorithms in conjunction with different work scheduling algorithms, determining the efficiency of the combination given different workloads.

Inadomi et al. [28] investigate methods of managing power in an HPC system using

RAPL, IBM BlueGene/Q, EMON, and PowerInsight to both measure and combat *power inhomogeneity.* Their study utilizes two of the same benchmarks DGEMM and STREAM, as well as several other benchmarks ranging CPU and Memory intensive, to I/O intensive. Using a model, they create a model making the assumption that power consumption for both CPU and DRAM is proportional to the CPU frequency. Using previously known information, such as the number of cores an application will be using, they calculate a maximum power budget. Ultimately, this work proves that RAPL can be used successfully to set a power ceiling.

Mars et al. [45] seek to address interference from co-located workloads. In particular, they focus on latency-sensitive workloads such as user facing applications. The Bubble-Up approach presented relies on measuring much pressure an application place on the memory subsystem and measuring how different levels of pressure affect the performance much an application. In our work, we co-locate tasks with the goal of utilizing components in an node as efficiently as possible. Interference between co-located tasks is not only bad for performance, it also has a detrimental effect on energy due to static power consumption. Our work quantifies the impact interference has on energy and power consumption.

Delimitrou et al. [13] present Quasar, a cluster management that seeks to increase resource utilization while allowing applications to maintain an adequate level of performance. Instead of allowing for reservation of resources, Quasar presents a model that allows users to specify performance requirements, leaving the decision engine to allocate the appropriate amount of resources. The work also explores the concept of interference, that is, when co-located workloads leads to a degraded performance compared to the performance of each workload run in isolation. Whereas the work presented by Delimitrou et al. focuses on achieving high utilization, our work focuses on the impact achieving such high utilization has on on power and energy consumption.

Delimitrou et al. [12] propose a system that quantifies the pressure applications place on hardware components. Leveraging this information they are able to understand how certain applications react to increased pressure being place on hardware and demonstrate how understanding, managing, and reducing application interference can improve efficiency and/or performance.

Vasan et al. [70] performed an in depth study of power consumption by servers in a data center. Using various metrics, they quantified if the power consumption of the servers was worth the work being carried out by each server. Further more, they provided insights into how servers could operate in a more efficient manner, such as allowing processors to enter into C-states [60] during idle times. Vasan et al. also introduce an empirical model for power consumption that correlates power and utilization. Our work builds upon the core idea of the objective value of each Watt consumed by the cluster in terms of carrying out work done. Our work differs in that we explore techniques that improve the work done by each Watt consumed through the use of work scheduling techniques as well as power capping.

Fan et al. [19] present a study of the power usage characteristics of large collections of servers for different classes of applications over a period of approximately six months. In this work they determine that even for well tuned applications, there can be a 7 to 16% difference between predicted power consumption and actual power consumption, making estimating power consumption difficult.

Chapter 3

Exploring the Design Space for Optimizations with Apache Aurora and Mesos

We address the peak-power collision problem using a policy-driven heuristic approach to the multi-dimensional bin packing problem. We use the DaCapo benchmarks[6] as workloads. DaCapo is a set of open source, real-world applications that exercise the various resources within a compute node. Our approach characterizes the power use of each benchmark on each node using fine-grained power profiles provided by Intel's Running Average Power Limit (RAPL)[10] counters via the Linux Powercapping framework [58]. We take the power profiling data for a given benchmark and node, and use that to engineer the job arrival time by potentially delaying it up to 3 seconds. This delay ensures that the power surge for two benchmarks do not coincide and also influences how Aurora allocates resources for each benchmark. We show the effect of two different bin packing policies, one that takes into account local power profile information and one that takes into consideration global power profile data. We evaluate how the staging of tasks to avoid peak power collisions also influences resource usage and energy consumption.

We make the following contributions in this chapter:

- We demonstrate how bin-packing a set of tasks can effectively reduce the peak-power usage and total energy usage while increasing the node utilization, when workloads are co-scheduled to be run using Mesos and Aurora.
- We show how our experimental pre-scheduler can inform application developers how their applications respond to peak power usage in a heterogeneous cloud environment.
- We demonstrate how Apache Mesos and Apache Aurora should be used so that application developers can express what they need from a cluster in terms of peak power use, and not just memory, disk, and CPU specifications.

3.1 Experimental Setup

Our experiments were conducted on the Binghamton University Cloud and Big Data Computing Laboratory's research cluster, which comprises the following components:

- 4 *Baseline CPU/RAM* nodes - Two 6 core, 12 thread Intel Xeon E5-2620 v3 @ 2.40GHz and 64 GB RAM
- 2 *Faster CPU/Baseline RAM* node - Two 8 core, 16 thread Intel Xeon E5-2640 v3 @ 2.60GHz and 64 GB RAM

- 2 *Fastest CPU/More RAM* node - Two 10 core, 20 thread Intel Xeon E5-2650 v3 @ 2.30GHz and 128 GB RAM

The workloads with which we assess performance were derived from the DaCapo Benchmark suite. Benchmarks were run inside Docker containers on the OpenJDK6 JVM. Performance of containers has been measured to be as at the same level, or better, than Virtual Machines [20]. Therefore, no performance degradation is expected compared to a Virtual Machine based experiment. Each workload possesses distinct characteristics[6] as listed in Table 3.1. Each node runs 64-bit Linux 4.2.0-18 and shares an NFS server. Apache Mesos 0.25.0 is deployed as the cluster manager, Apache Aurora 0.11.0 as the scheduler, and Docker 1.9.1 as the container technology. Performance Co-Pilot collects metrics for all nodes in the cluster. These metrics include energy measurements from RAPL counters, and various statistics about CPU and memory usage from the worker nodes. An Ansible playbook is deployed to limit power consumption by the nodes in the cluster using the Linux Powercapping framework[58].

The DaCapo benchmarks `tradebeans`, `tradesoap`, and `tomcat` simulate cloud workloads with memory and network intensive components. The remaining benchmarks simulate diverse types of workloads varying from highly parallel workloads to highly serial tasks.

Power Throttling (Power Capping): To control the power usage by CPUs in our cluster, we used the Linux Powercapping framework [58]. The Powercapping framework takes advantage of Intel's RAPL. RAPL creates a power estimate based on a highly accurate software model [60]. The power estimate value determines the `P`-state [3] at which the processor must operate to make the best effort to meet a

avrora	Multithreaded AVR microcontrollers simulator
batik	Scalable Vector Graphics generator, limited concurrency
eclipse	Eclipse IDE perf. tests, mixed concurrency
fop	Multithreaded PDF generator from XSL-FO
h2	Multithreaded, in-memory benchmarks
jython	Python benchmark, limited concurrency
luindex	Documents indexer, limited concurrency
lusearch	Multithreaded keyword finder
pmd	Multithreaded Java source code analysis
sunflow	Multithreaded Raytracer
tomcat	Multithreaded server
tradebeans	Multithreaded daytrader benchmark. Uses Java Beans, in-memory database, and GERONIMO
tradesoap	Multithreaded daytrader benchmark. Uses SOAP, in-memory database, and GERONIMO
xalan	Multithreaded XML to HTML converter.

Table 3.1: List of benchmarks in the DaCapo Suite

power budget. [10] For this set of experiments, we set the power-cap at 50% of the TDP for each node. This percentage was determined to be the most appropriate through experimentation.

Aurora Job creation: To orchestrate the experiments on the Mesos cluster we generated several Aurora job configurations, including multiple versions of the same benchmark constrained to only run on a specified node. Our job launcher submits these jobs through Aurora's client-side application, which converts the Domain Specific Language job configuration into an Apache Thrift Remote Procedural Call made to the Aurora Scheduler. The job then progresses through the lifecycle of an Aurora job described in Section 1.2 .

Experimental workflow: The Bin-Packer generates a set of bins, each containing a set of tasks. This information is passed to a job launcher. The job launcher generates the appropriate Aurora job configuration that results in a specific task being launched. The set of tasks is submitted in groups determined by the start and end of each bin with a fixed delay between bins. We set a configurable delay value at three seconds because most of the tasks take approximately three seconds to experience a power spike and subsequently drop power consumption to its average consumption.

Profiling: We run the benchmarks several times on each node and use the information from each run to determine valuable characteristics about each benchmark including when it experienced a power surge, CPU utilization, and makespan.

3.2 Max Peak First Bin Packing (MPF-BP)

MPF-BP is a bin-packing algorithm. A sorted list of all the peaks from the profiling stage is created. The size of each bin is determined by the thermal design power (TDP) of a given node. MPF-BP fits peaks into a bin such that the resulting set of tasks in each bin is $\sum_{i=1}^{n} PeakPower(Task_i) \leq TDP$ where n is the number of tasks in the bin. When a bin reaches its maximum capacity or there are no more tasks in the queue that fit in the bin, a new bin is created. These steps are repeated until all the workloads have been placed inside a bin.

Power Usage on 3 different nodes with MPF-BP: We conducted experiments to study the peak power usage, energy consumption, and memory utilization when MPF-BP is applied to three different nodes - baseline, faster, and fastest. The results for the baseline node are presented in Figures 3.1 to 3.3 and described below. The Figures for *faster* and *fastest* display similar characteristics and have therefore been omitted, however, the results are summarized below in this subsection.

Algorithm 1 MPF-BP

```
$n \leftarrow 0$
$Bin_n.capacity \leftarrow CPU_{TDP}$
$Q \leftarrow$ Max peaks in non-decreasing order
while Q is not empty do
    $Task \leftarrow Q.pop()$
    if $Bin_n.capacity - Task_{PeakPow} \geq 0$ then
        $Bin_n.capacity \leftarrow Bin_n.capacity - Task_{PeakPow}$
        $Bin_n.insert(Task)$
    else
        $Q.push(Task)$
        $n \leftarrow n + 1$
        $Bin_n.capacity \leftarrow CPU_{TDP}$
    end if
end while
```

Baseline node power characteristics: As shown in Figure 3.1, the max peak power used by the node is about the same when compared to the default configuration.

Figure 3.2 shows the memory usage for the three cases. Compared to the default run, the MPF-BP optimized and uncapped run achieved a 48% decrease in total memory usage, while the power-capped-bin-packed achieved a 31% total memory use decrease compared to the default run.

Figure 3.3 shows that when the node's power is throttled by 50%, the energy expenditure is reduced by 64% compared to the baseline run, and is about 4% more efficient than the bin-packed optimized case.

Faster node power characteristics. On the faster node, (Figure not shown) peak power is not substantially different from the runs containing bin-packing and power-capping. However, the frequency with which peaks are reached is reduced, from four large power spikes to three large spikes in the bin-packed run. Memory utilization

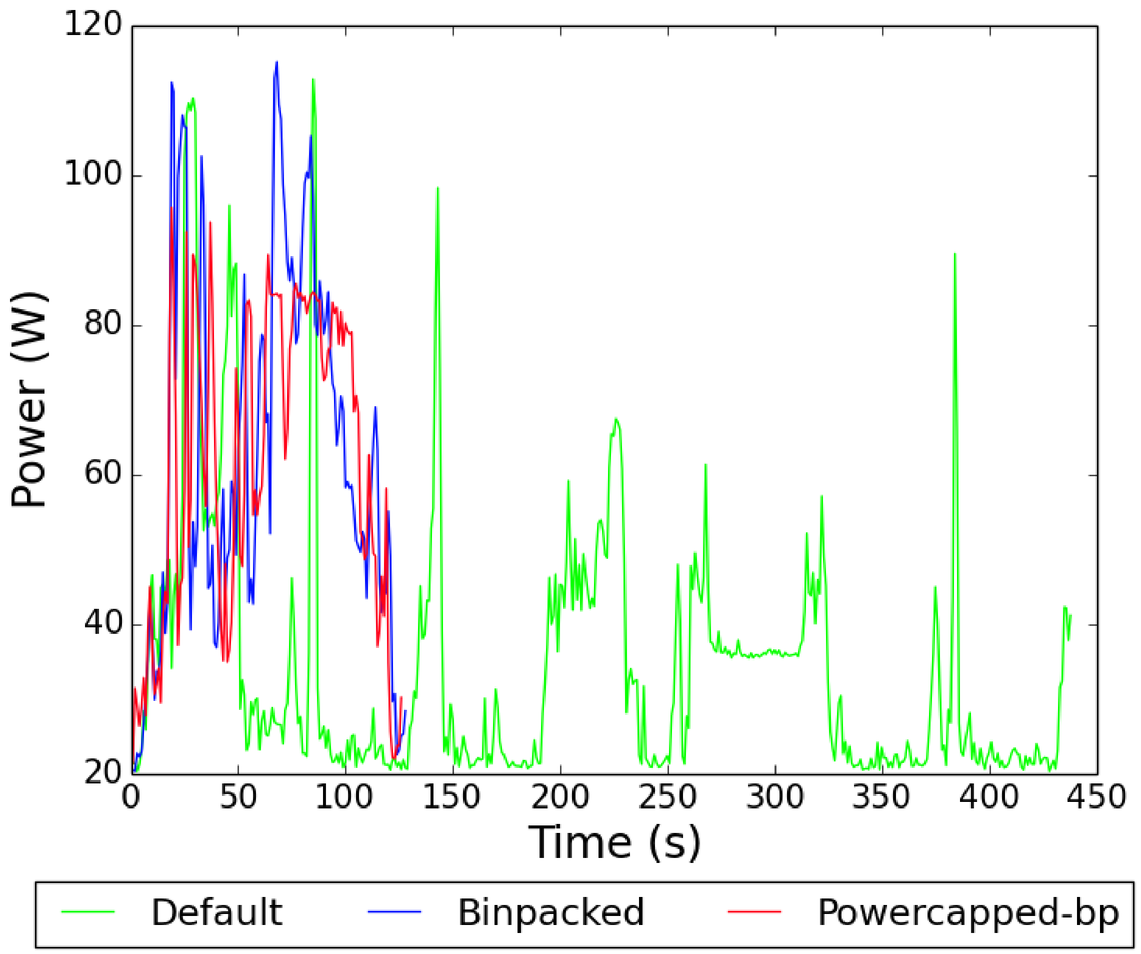

Figure 3.1: Baseline node power consumption. Outlines the effects of power capping the baseline node. The max peak power is reduced using a power-cap of 50% of the total TDP of the node

data shows that compared to the default configuration, the MPF-BP optimized and uncapped run achieved a 131% decrease in total memory usage, while the power-capped-bin-packed approach achieved a 155% total memory use decrease compared to the default run.

Fastest node power characteristics: On the fastest node (Figure not shown), the difference in peak power between the default run and the MPF-BP optimized run

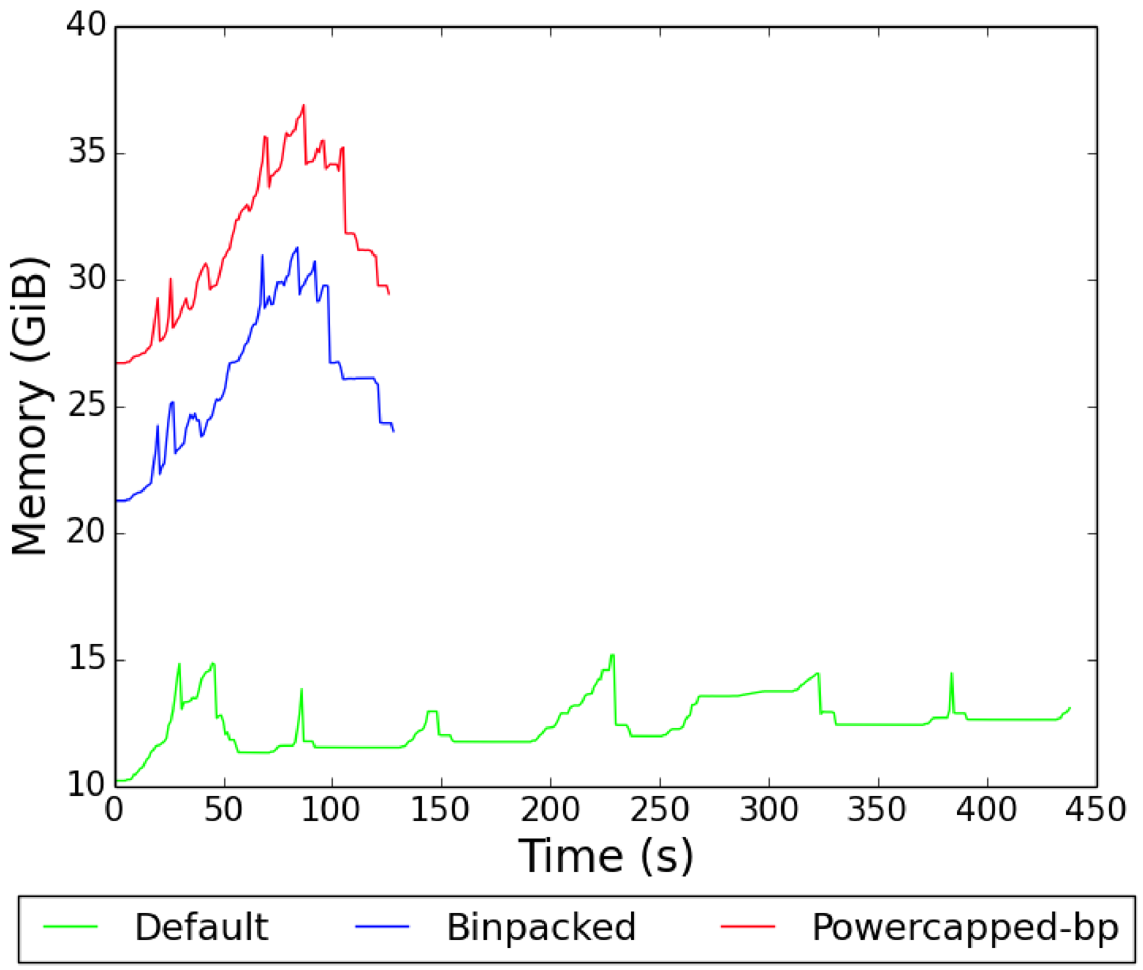

Figure 3.2: Baseline node Memory utilization. This figure shows that compared to the default run, the MPF-BP optimized and uncapped run achieved a 48% increase in total memory utilization, while the power-capped-bin-packed achieved a 31% total memory use increase compared to the default run.

is less than the default case by 4%. For memory utilization, compared to the default configuration, the MPF-BP optimized and uncapped run achieved a difference of 77% decrease in total memory utilization, while the power-capped-bin-packed run achieved a difference of 78% total memory use decrease compared to the default run. The total

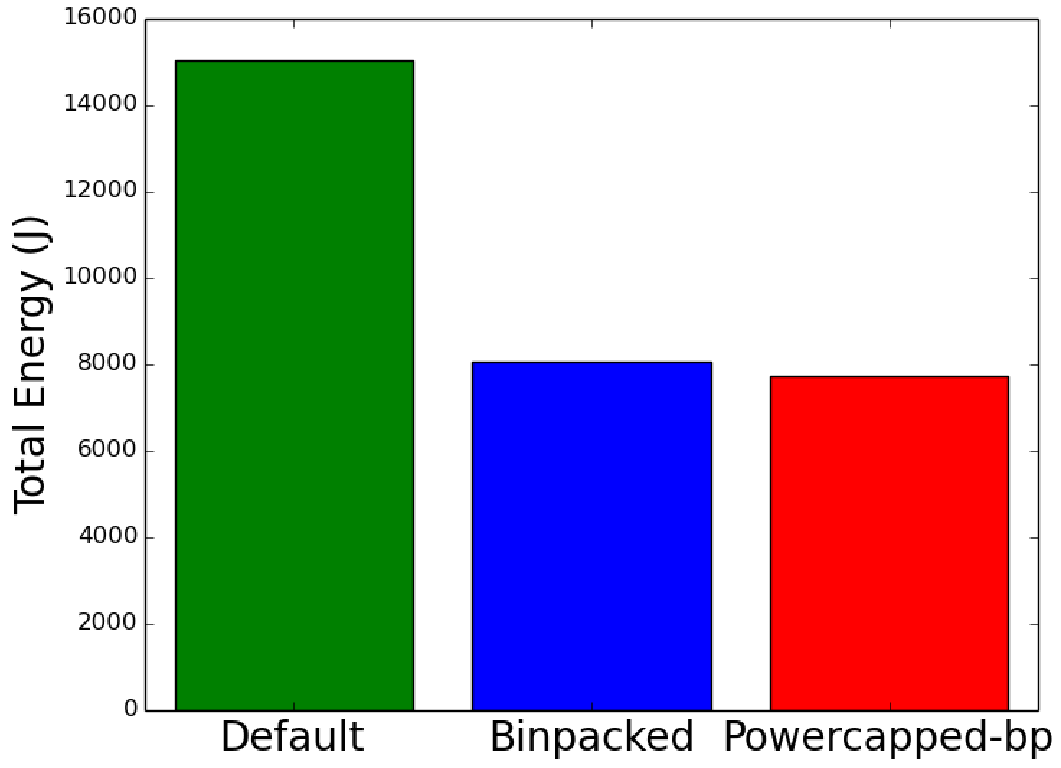

Figure 3.3: Baseline node Energy consumption.

energy expenditure improved in the bin-pack only run by 28% compared to the bin-packed, power-capped run by 42% as compared to the default run. An improvement in both of these areas is only experienced when the MPF-BP is used with a power-cap of 50%. The max peak power is decreased by 27% compared to the bin-pack only run, and 23% compared to the default run. The energy expenditure sees an improvement of 22% over the bin-pack only run and 54% over the default run.

3.3 Max Average Peak Bin Packing (MAP-BP)

MAP-BP generates a sorted list of means for all workloads based on the power consumed by each particular workload. The median is calculated for the set containing the mean power used by each benchmark. A bin is packed with tasks that have a higher peak than the median of the set. The steps are repeated once a bin reaches capacity until all tasks have been placed in a bin. Once again, the aggregate sum of the max power of the set of tasks is not allowed to exceed the TDP of the node they will be scheduled on.

Power Usage on 3 different nodes with MAP-BP: We conducted experiments to study the peak power usage, energy consumption, and memory utilization when MAP-BP is applied to three different nodes - baseline, faster, and fastest. Again, the results for just the baseline node are presented in Figures 3.4 to 3.6. The results for all three nodes are described below.

Baseline node power characteristics. The bin-packed only run in Figure 3.4 shows similar power peaks to those observed in Figure 3.1. The bin-packed only run for MPF-BP experiences a similar max peak as the bin-packed only MAP-BP run. The max peak of the power-capped MPF-BP policy reaches above 100 watts while the power-capped MAP-BP policy is below the same threshold. The bin-packed only run reaches a max peak of around 110 Watts while the power-capped-bin-packed max peak only reaches about 90 Watts.

Faster node power characteristics. In terms of peak power, the bin-packed-only run has a peak of 140 Watts, which is 20% higher than any peak in Figure 3.1. Furthermore, its peaks are 24% higher than the baseline. The power-capped-bin-packed case on the other hand sees a similar improvement to MPF-BP with only one

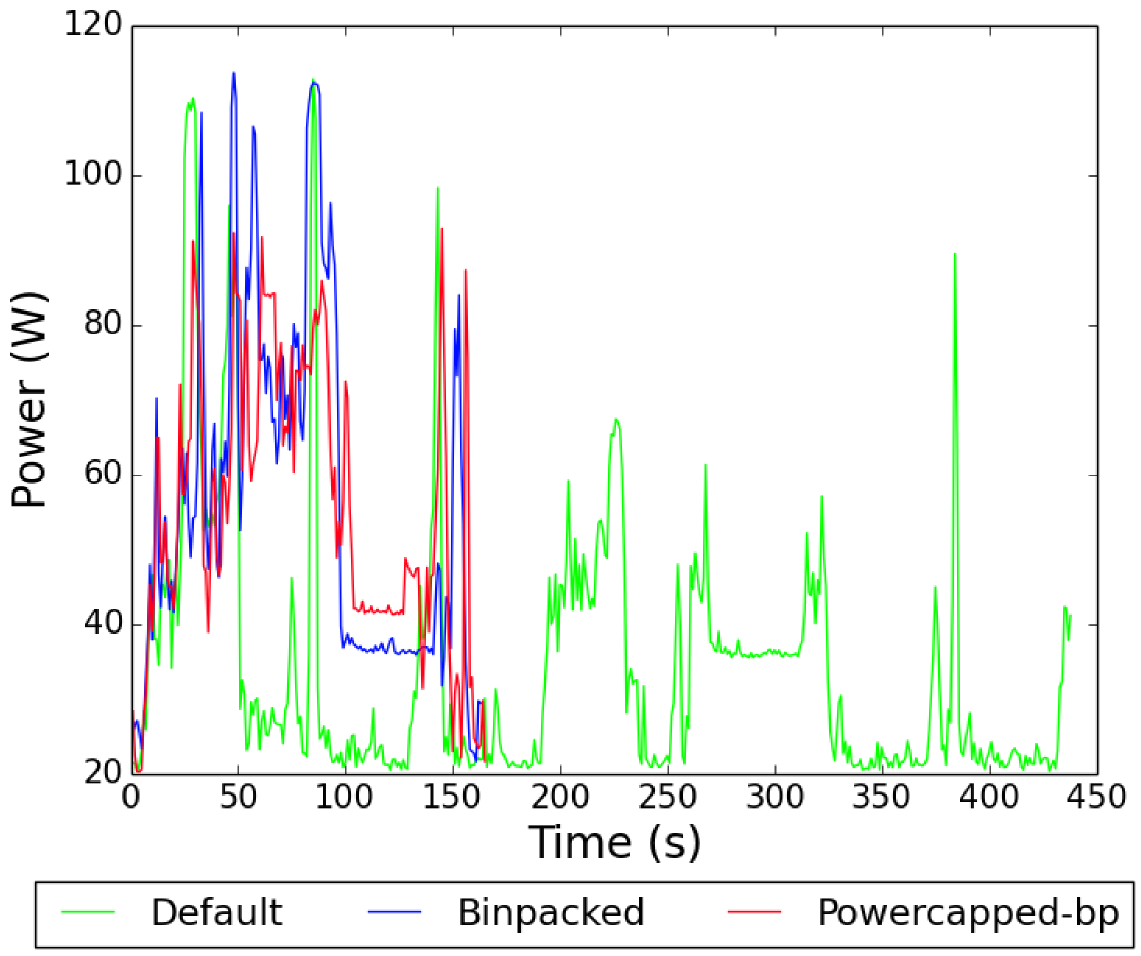

Figure 3.4: Baseline node Power consumption using MAP. The bin-packed only run shows similar power peaks to those observed in Figure 3.1. The most noticeable difference is the decrease in consumption around 100 seconds and a late spike at around 150 seconds. The power-capped-bin-packed run displays similar characteristic with lower peaks by 90% when compared to the baseline and 1% compared to the optimized run.

peak crossing the 100 Watt threshold. Energy usage for the bin-packed-only run sees similar results, with MAP-BP based run incurring 6% more energy consumption. The power-throttled-bin-packed run sees a 15% jump in energy consumption compared to its MPF-BP counterpart. Memory utilization shows that compared to the default

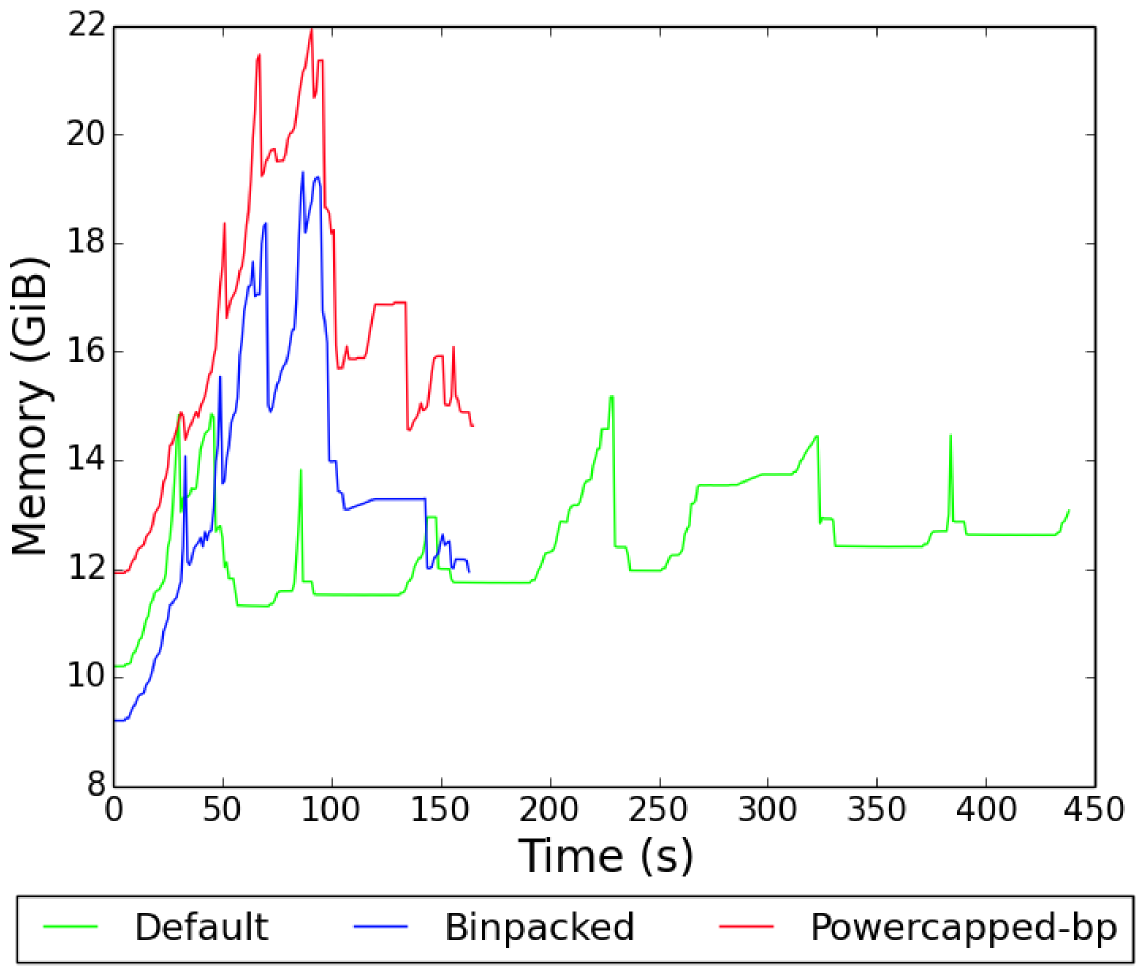

Figure 3.5: Baseline node Memory Utilization using MAP. Memory utilization, shown in Figure 3.5, shows that compared to the default run, the MPF-BP optimized and uncapped run achieved a 85% increase in total memory utilization, while the power-capped-bin-packed achieved a 67% total memory use increase compared to the default run.

run, the MPF-BP optimized and uncapped run achieved a 148% increase in total memory utilization, while the power-capped-bin-packed approach achieved a 127% total memory use increase compared to the default run.

Fastest node power characteristics. The peak power consumption achieved by the bin-packed only run is similar to that exhibited in MPF-BP, with no change

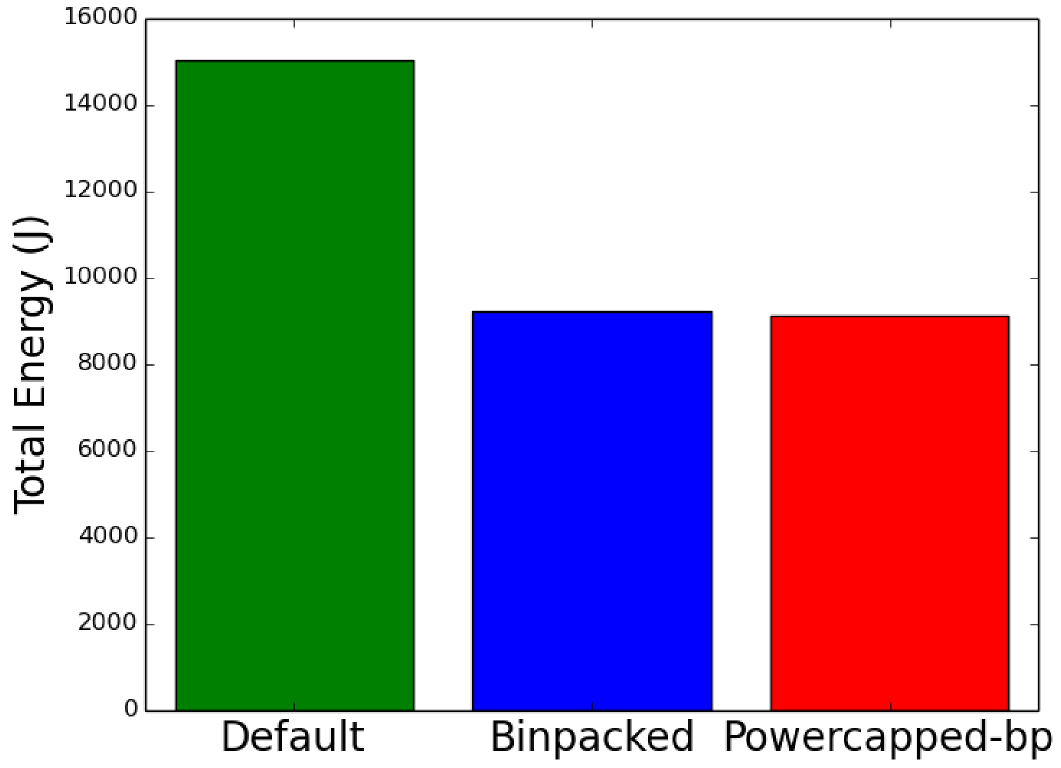

Figure 3.6: Baseline node Energy consumption using MAP. MAP-BP, Baseline Node using the MAP-BP policy. The energy utilization compared to the MPF-BP is 14% higher for the bin-packed-only run and 17% higher for the power-capped-bin-packed run.

in max peak. The power-capped-bin-packed run, however, reaches a max peak that is 11% less compared to its MPF-BP counterpart. The bin-packed-only version has a decrease of 11% compared to the MPF-BP bin-packed only run. Total memory usage shows that compared to the default run, the MPF-BP optimized and uncapped run achieved a 70% increase in total memory utilization, while the power-capped-bin-packed achieved a 48% total memory use increase compared to the default run.

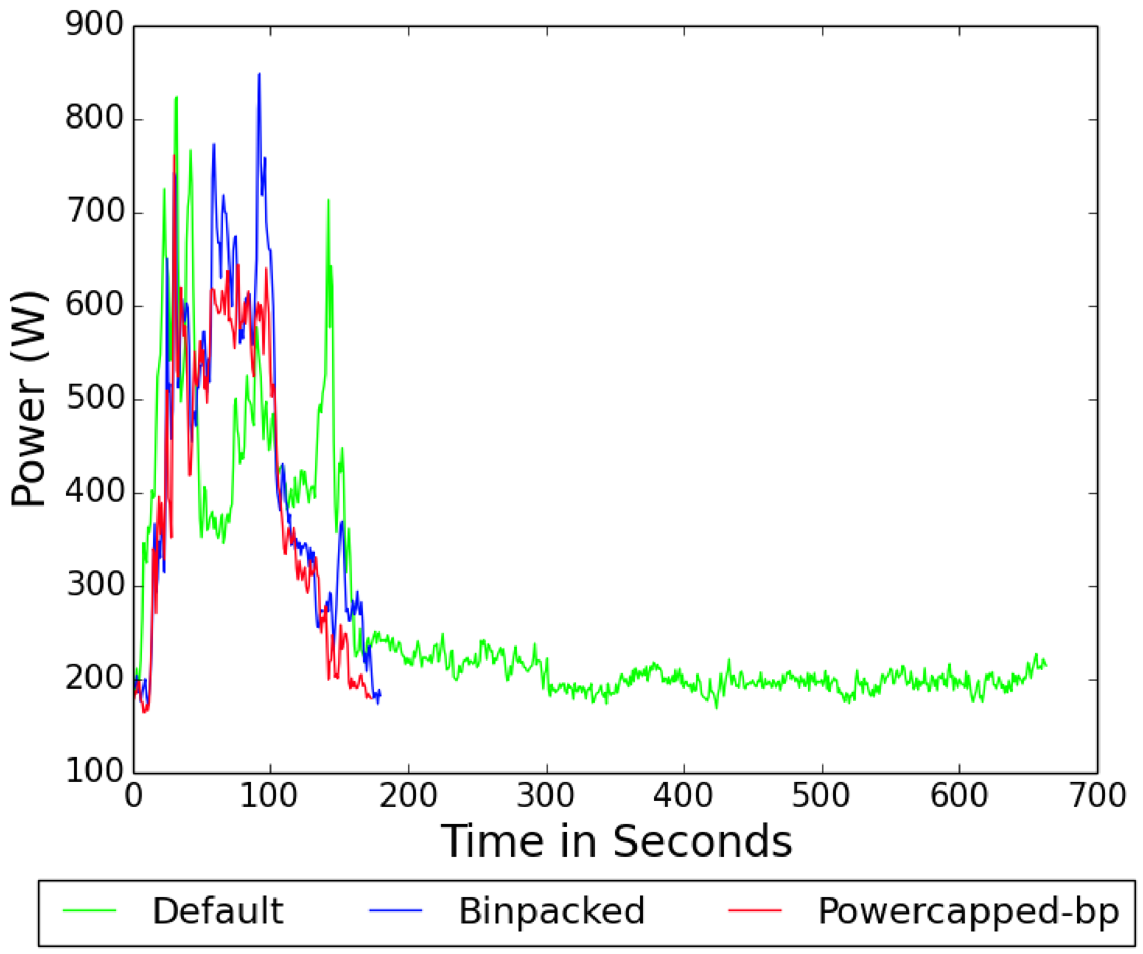

Figure 3.7: Power Utilization of cluster using MPF-BP. The power usage for the entire cluster shows a similar trend to the one seen in Figure 3.1 The max power peak for the default run, however, is 3% smaller than the max peak of the cluster running the MPF-BP. The max peak for the power-capped-bin-packed run shows an improvement over the other two runs by 8% against the default run and 11% against the power-capped-bin-packed run.

3.4 MPF-BP for the Entire Cluster

Figures 3.7 to 3.9 show the MPF-BP policy applied to the entire cluster. Each node was responsible for bin packing its own set of tasks. Taking into account the power usage for the entire cluster, the results show a similar trend to the one seen in Figure

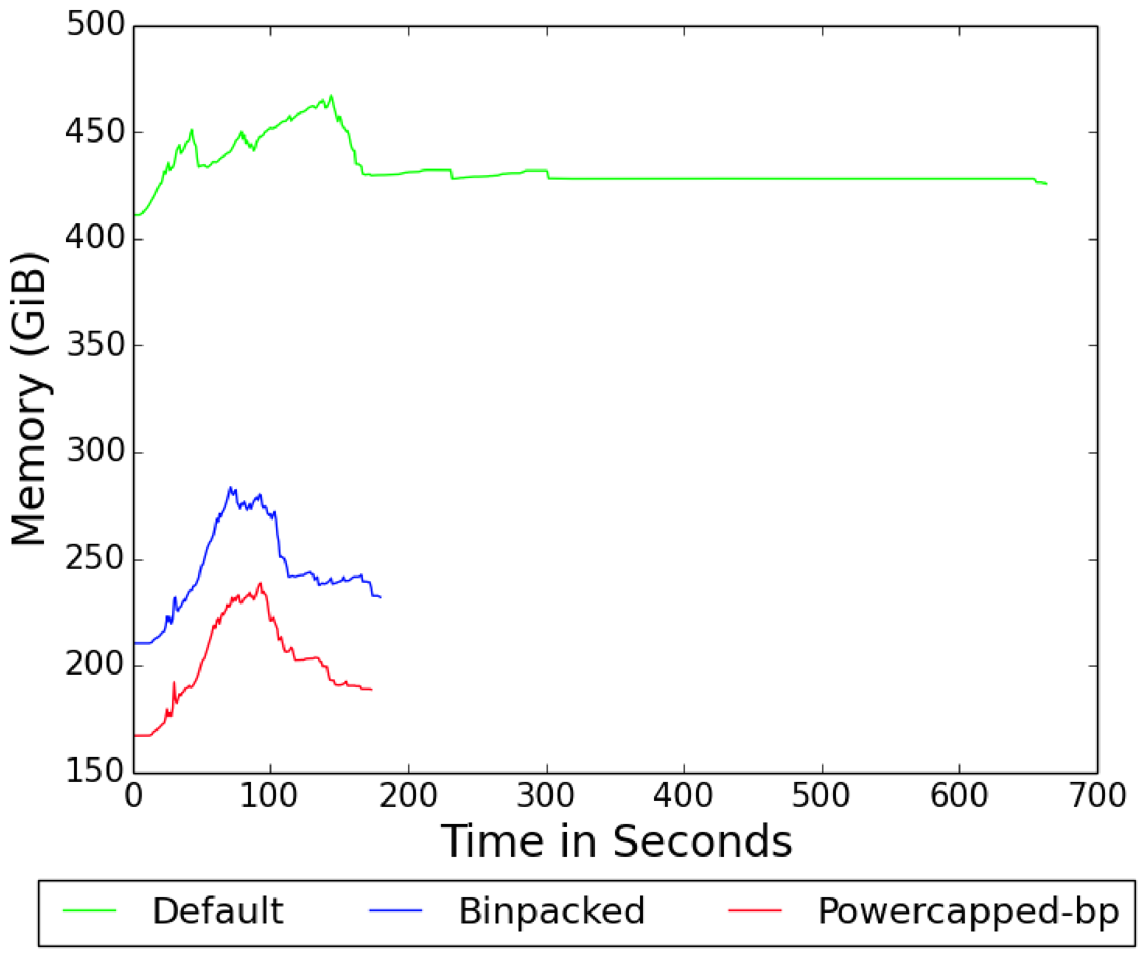

Figure 3.8: Memory Utilization of cluster using MPF-BP.Memory utilization is, at its highest, 147% higher than the bin-packed only run and 157% higher than the power-capped-bin-packed run. The default run expends 77% more energy than the bin-packed only run and 86% more energy than the power-capped-bin-packed run.

3.1, The max peak for the power-capped-bin-packed run shows an improvement over the other two runs by 8% against the default version, and 11% against the bin-packed only version. Memory utilization, at its max, was 147% higher than the bin-packed only run and 157% higher than the power-capped-bin-packed run. Finally, similar to results shown in Figure 3.9, the default run expends 77% more energy than the

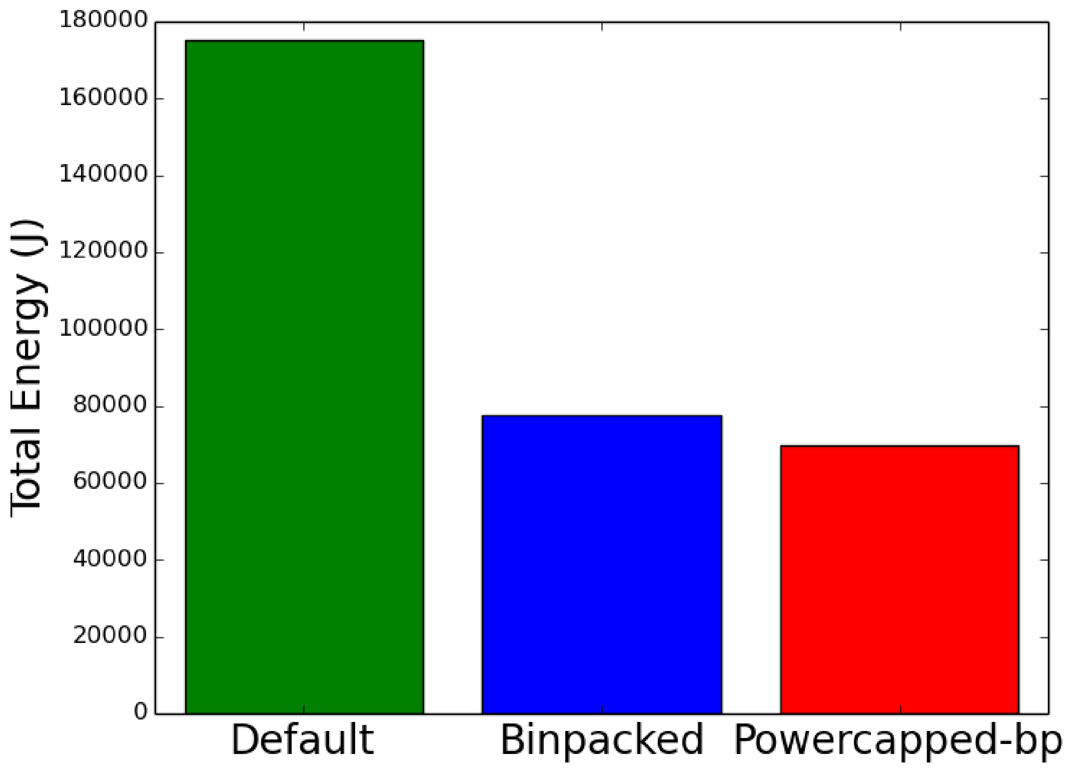

Figure 3.9: Energy Consumption of cluster using MPF-BP.

bin-packed only run, and 86% more energy than the power-capped-bin-packed run.

3.5 MAP-BP for the Entire Cluster

The power trend for the bin-packed only and power-capped-bin-packed follow similar patterns, with a 11% difference in peaks. The peak power for the bin-packed only approach improves upon the cluster-wide MPF-BP bin-packed only max peak, shown in Figure 3.7, by 10% while the power-capped-bin-packed version also improves upon its MPF-BP cluster-wide counterpart in Figure 3.7 by 10%. The energy utilization

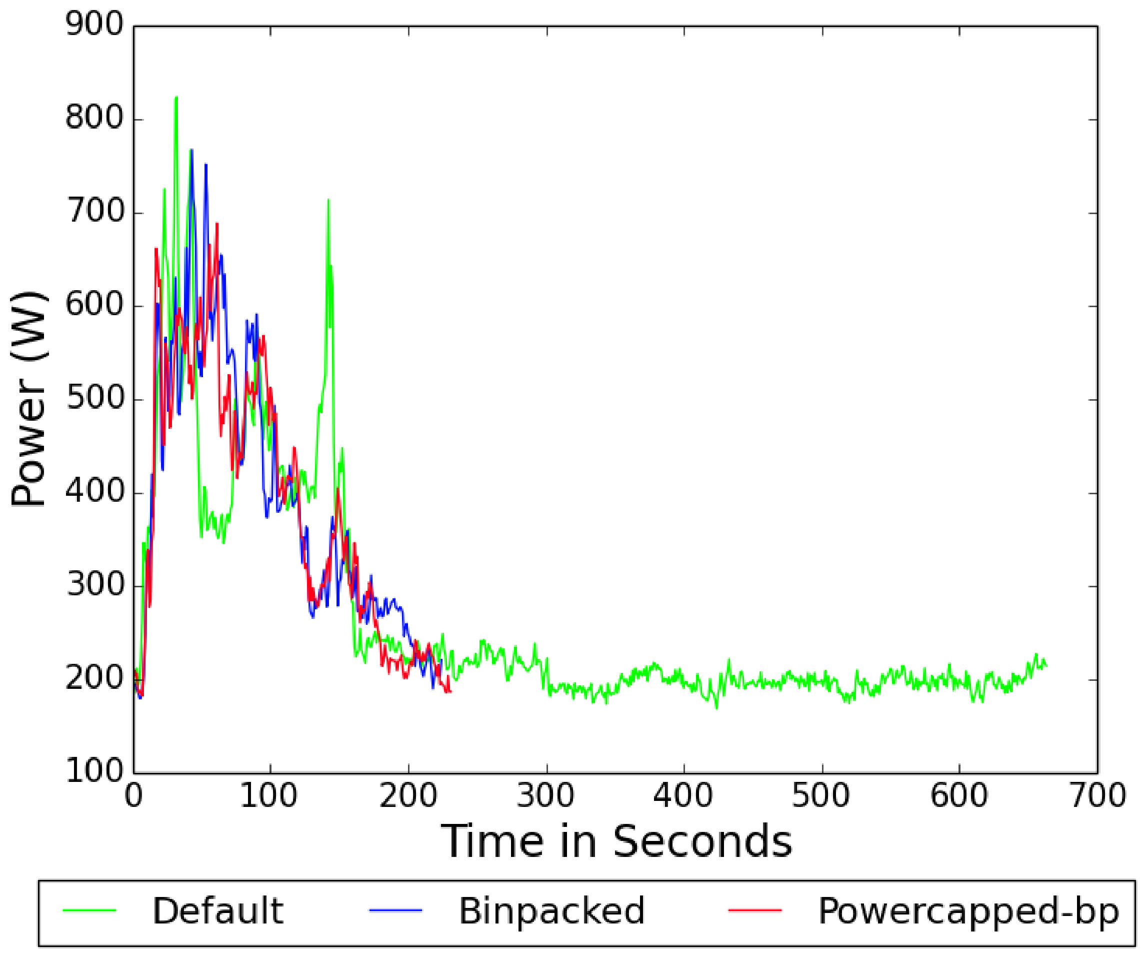

Figure 3.10: Power consumption of cluster using MAP-PB.The power trend for the bin-packed only and power-capped-bin-packed follow similar patterns, with the difference in peaks being 11%. The peak power for the bin-packed only improves upon the bin-packed only max peak in Figure 3.7 by 10% while the power-capped-bin-packed improves upon the power-capped-bin-packed max peak in Figure 3.7 by 10%.

compared to MPF-BP algorithm is similar to the bin-packed only version. The power-capped-bin-packed version has a 23% increase compared to the power-capped-bin-packed version run in Figure 3.9.

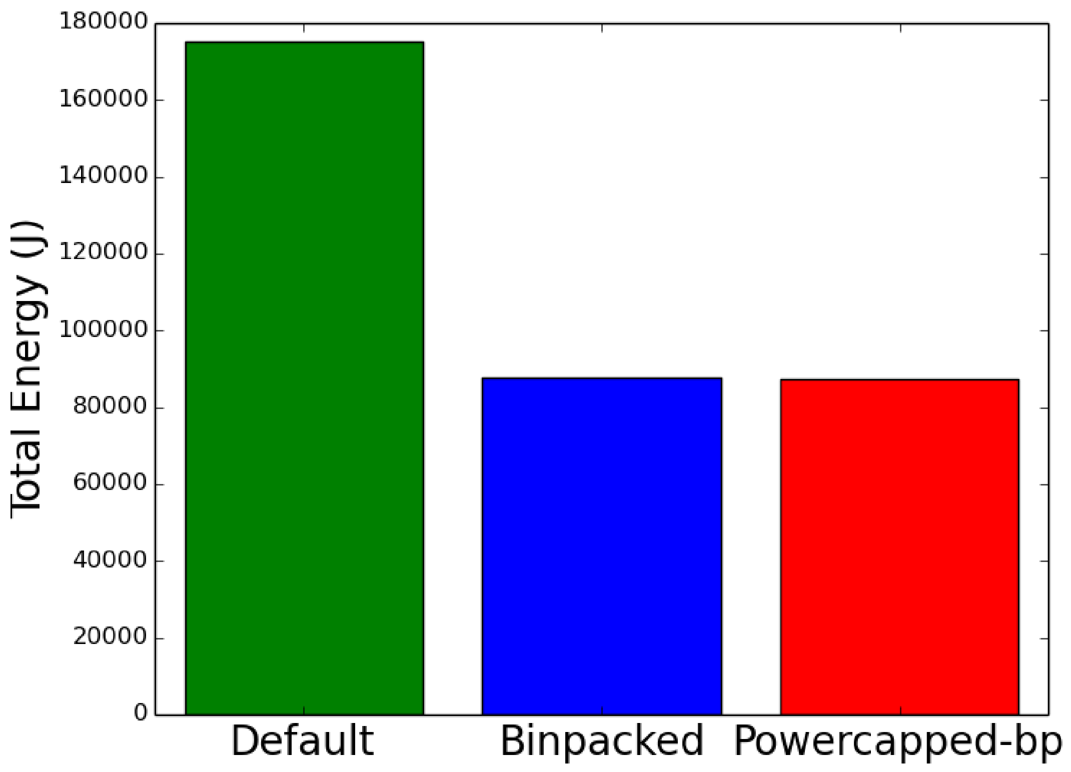

Figure 3.11: Energy consumption of cluster using MAP-BP. The energy utilization is similar to the MPF-BP policy.

3.6 Analysis of our Policies on Aurora and Mesos

The improvements for both the policies can be attributed to the default task scheduling mechanism that exists in Mesos and Aurora. The default mechanism does not take into account the power profiles of the nodes for a given benchmark. Our policies ensure that tasks are scheduled such that multiple tasks do not experience peak power at the same time. Mesos recycles non-reserved resources from an offer as a new, smaller resource offer [25]. The improvements experienced by our bin-packing policies can be further attributed to Apache Aurora's offer caching mechanism and

Mesos' resource recycling procedure reacting positively to the configurable 3 second delay we have inserted between tasks. Aurora's first match scheduling is now forced to follow the scheduling order our policies have enforced instead of a generic policy it uses by default that is based on the jobs that are in the queue and which resource offers have been received and cached. Mesos and Aurora are top level Apache projects that are subject to active contributions by the open source community. We developed a framework to evaluate and analyze different policies for using these tools, off the shelf, without making any changes to Mesos 0.25.0 nor Aurora 0.11.0.

3.7 Conclusion

Our policy driven approach is highly effective for use with Apache Mesos and Aurora.

- The order in which tasks are co-scheduled in Mesos and Aurora has a significant impact on energy usage, resource utilization, and peak power usage.
- The MPF-BP policy achieves 8% reduction in peak power usage compared to the default run of Mesos and Aurora in a cluster. Additionally, it provides a gain of 86% in energy savings.
- The power-capped-bin-packed MAP-BP policy, compared to the power-capped-bin-packed MPF-BP policy, achieves 10% reduction in peak power usage. However, it suffers an increase of 95% in energy consumption in comparison to the power-capped-bin-packed MPF-BP policy.
- Inserting delays between sets of job submission mitigates penalties created by large resource offers and small tasks.

- Based on results, prioritizing the tasks with the highest energy peaks (MPF-BP) results in the favorable scenarios compared to MAP-BP and default. It must be noted, however, that MPF-BP has the potential for starving tasks with the lowest peak power. Additionally, these approaches require a power-profile which may not always be available and the study is limited in scope to a only a few applications.

Chapter 4

Electron: Towards Efficient Co-Scheduling on Heterogeneous Clusters with Apache Mesos

The landscape for massive workload execution is dominated in the industry by large-scale data centers running different cloud software management tools to efficiently manage the large infrastructure. With the recent availability of virtualized clouds for production use, a similar shift has started for science and academic applications such as JetStream [67] and Chameleon [9]. Among the software management tools being deployed in data centers and clouds, Apache Mesos [25], with its *data center operating system*, has gained significant traction in the industry. A key tenet of successfully using data centers to their full potential is ensuring that the utilization of the components remains high. High utilization of resources, however, often bears a cost in terms of energy and power consumption. Moreover, unchecked peak power draws can lead to increased costs for both data centers and power suppliers[5]. Thus, our focus is on reducing peak power draws and energy consumption without having a large impact on makespan and resource utilization.

Marathon [44] and Aurora [4] are the widely used frameworks that interact with Mesos to submit jobs, negotiate resource offers, and receive feedback on the status of jobs. While both Marathon and Aurora support container orchestration, via Docker [48] and Mesos containers, Marathon is only suited for long-running services that need *init* style functionality such that the services are monitored and restarted whenever needed.

Mesos, Aurora, Marathon, and other tools in this space are designed to allow co-scheduling of tasks on nodes, with isolation between tasks provided by containerization. When tasks are co-scheduled, the power they draw from the node is dependent on (a) how efficiently they use the hardware resources, and (b) whether the peak power draws of different tasks coincide. It is known that optimal co-scheduling of applications to minimize peak power usage can be reduced to the NP-Hard multi-dimensional Bin Packing problem [46]. To accommodate this, our approach uses heuristics that can prioritize different policies such as reducing peak power usage, reducing energy consumption, and improving resource utilization while maintaining similar execution time to non-power-aware runs. We quantify the improvements in power and energy usage along with the impact that these policies have on the makespan and resource utilization for a set of workloads run on Apache Mesos.

By design, Aurora uses a First Fit policy exclusively, and was not designed to be energy conscious. We therefore designed and developed our own framework, *Electron*, with similar characteristics to Aurora, to serve as a proof of concept that a framework that is capable of combining different scheduling policies with power capping strategies can be leveraged for power aware clouds. Electron is designed as a lightweight, configurable framework, which can be used in conjunction with built-in power capping policies to reduce the peak power and/or energy usage of co-scheduled tasks.

We make the following contributions in this paper:

- We have developed a Mesos framework, Electron, to show how a combination of scheduling policies and power capping strategies leads to lower peak power draws and/or lower energy consumption.
- We quantify the effect of Electron on resource utilization (CPU and memory) and makespan.
- We demonstrate the following fundamental strategies for consuming Mesos offers: First Fit and Bin Packing. While we demonstrate the use of these select policies to show the efficacy and viability of Electron, several other policies proposed in the literature can also be applied for use with Mesos via our framework.
- We quantify the impact of using RAPL to cap a cluster while using First Fit or Bin Packing as the Mesos offer selecting strategies. We introduce a dynamic capping strategy, *Extrema*, to mitigate the impact of power capping on makespan.

4.1 A new Framework

Aurora allows for mass scheduling of tasks on Mesos controlled resources. However, Aurora's scope is limited to only allowing users to schedule ad-hoc jobs, service jobs,

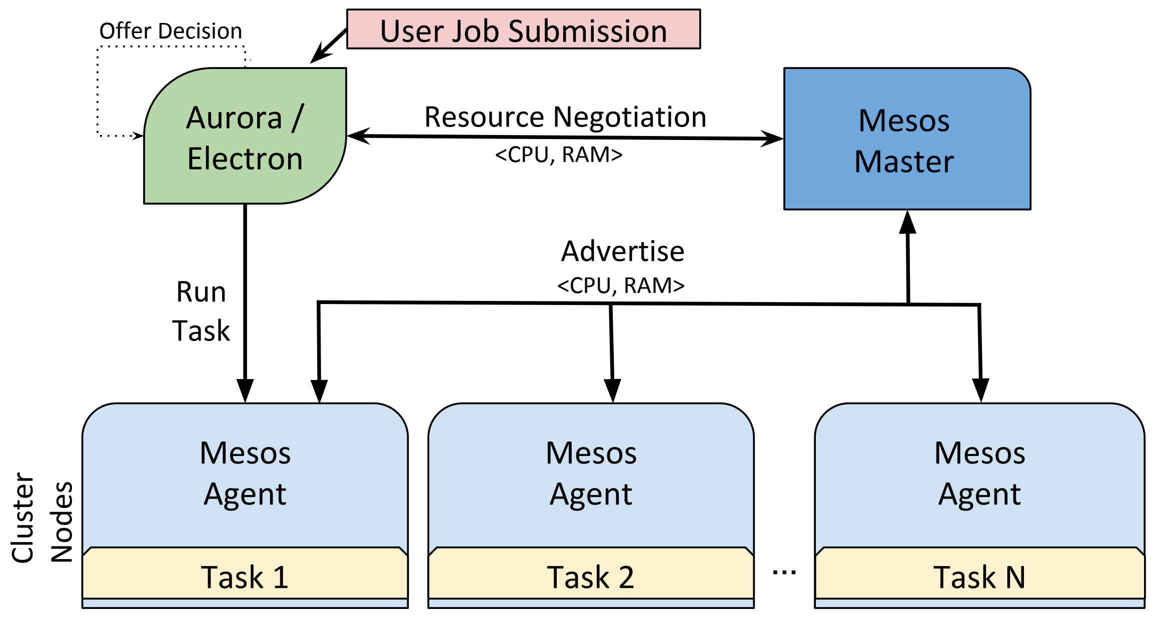

Figure 4.1: Architecture of Mesos and Aurora/Electron. Note that Electron has the same position in the architecture as Aurora.

and cron jobs based on three constraints CPU, Memory, and Disk. The end user has no input on how Aurora should make use of Mesos resource offers. Important details like how the tasks queue should be ordered and how many tasks to schedule on an offer are not controllable by the end user. To consume Mesos resource offers, Aurora has been fitted with a First Fit algorithm which makes it very simple to implement more complex features like preemption. It became immediately clear that being bound to a single Mesos resource offer consumption approach was severely limiting. Thus, in order to have full control over how tasks would be scheduled on the cluster based on Mesos resources offers and to explore different ways of consuming offers and to quantify the impact offer consumption and queue ordering has on resource utilization, power, and energy consumption, a new framework, Electron, was created.

4.2 Consuming Mesos Offers

Mesos sends each registered framework a coarse-grained list of resource offers. Frameworks are responsible for deciding how to consume these offers. There are two fundamental ways for frameworks to make use of the resources present in a Mesos offer: First Fit and Bin Packing. It should be noted that when resources available in an offer are not consumed in their entirety by a framework, any leftover resources are recycled by Mesos and are advertised as part of a new offer.

4.2.1 First Fit

Mesos sends each registered framework a coarse-grained list of resource offers. Frameworks are responsible for deciding how to consume these offers. There are two fundamental ways for frameworks to make use of the resources present in a Mesos offer: First Fit and Bin Packing. It should be noted that when resources available in an offer are not consumed in their entirety by a framework, any leftover resources are recycled by Mesos and are advertised as part of a new offer.

4.2.2 Bin Packing

This approach attempts to fit as many tasks as possible from the queue of tasks waiting to be scheduled into a single offer before moving to the next. If a task does not fit in the current offer, the next task in the queue is evaluated for fitting. If no task fits in an offer, that offer is rejected. The type of queue used for storing tasks, which are waiting to be scheduled, remains up to the framework designer. For these experiments, we used a priority queue keyed by the expected power usage. In this instance, we calculated this value by cataloging the max peak power over ten runs for each benchmark and using the median of the max peaks as the expected power usage value. This value is meant to represent the worst case scenario of power draw for each benchmark. This value only influences the order in which benchmarks are attempted to be scheduled and we acknowledge that there are other statistically valid values that are also suitable choices.

4.3 Measuring and Limiting Power Consumption

In our experiments, we used RAPL to monitor and limit power consumption. RAPL provides us with the ability to deploy our system on any cluster which is powered by Intel hardware based on the Sandy Bridge[1] architecture or newer. This provides us with improved flexibility when scaling our system, compared to placing power meters on nodes in existing clusters due to the usual lack of physical access to most data centers. Furthermore, although RAPL readings are estimates, Khan et al. have shown that they are accurate enough to extrapolate wall power usage [35], while Petoumenos et al. have quantified RAPL's ability to cap power consumption successfully[56]. On servers, RAPL readings are available for CPU and DRAM power consumption. In our experiments, we aggregate both of these readings to observe and record power consumption.

4.3.1 Static Power Capping

Static capping sets a loose upper bound on power consumption by setting each node in the cluster to a fixed fraction of their Thermal Design Power (TDP). However, static

[1]Sandy Bridge architecture was released in 2011.

capping neither considers the power usage trends of the cluster, nor the workloads that are executing on the nodes in the cluster. Thus, for power intensive benchmarks, a static cap leads to an increase in makespan as RAPL reduces a processor's clock speed in an effort to remain within a power budget. Since a static cap is a one size fits all capping strategy, it is difficult to determine a generic cluster-wide static cap value.

For our set of experiments, through taking into account the characteristics of our cluster and our workloads and through experimentation, we determined that setting a cap equal to 50% of each node's TDP saw the most beneficial power and energy reductions versus increased makespan trade off.

4.3.2 Extrema Dynamic Power Capping

In order to address some of the shortcomings of the static capping policy, we designed a dynamic capping policy that is able to make smarter trade-offs between makespan and power consumption.

The Extrema capping algorithm, shown in Algorithm 2, is designed to react to power trends in the cluster and restrain the power consumption of the cluster within a power envelope. This is done by monitoring and reacting to the power usage trend of the cluster and preventing it from crossing defined high and low thresholds. The Average Historical Power (AHP) value is compared against a *high threshold* and a *low threshold*. If the cluster's AHP exceeds the *high threshold*, the node that used the highest amount of power is capped. Otherwise, if the cluster's AHP is below the *low threshold*, a node that was last capped is uncapped. Ordering the capping of nodes in this way eases the transition to a fully uncapped cluster. For this set of experiments, and taking into account the results of our static capping experiments, we picked a 50% capping value for nodes chosen to be capped. Expectedly, Extrema's efficacy is dependent on being able to determine accurate high and low thresholds, which vary for each application class and scheduling policy.

Algorithm 2 Extrema Dynamic Capping algorithm

```
1: procedure EXTREMA_CAP($Threshold$)
2:     $ClusterAvg \leftarrow Avg_{running}(ClusterPower)$
3:     $CappedNodes \leftarrow Stack()$
4:     if ClusterAvg > Threshold.Hi then
5:         $Victims \leftarrow Sort_{non-inc}(AvgPowerNode[...])$
6:         for Victim in Victims do
7:             if Victim not in CappedNodes then
8:                 Cap(Victim)
9:                 CappedNodes.Push(Victim)
10:                break
11:            end if
12:        end for
13:    end if
14:    if ClusterAvg < Threshold.Low then
15:        Uncap(CappedNodes.Pop())
16:    end if
17: end procedure
```

4.4 Experimental setup

Our experiments were conducted on the Stratos cluster in the Binghamton University Cloud and Big Data Computing Laboratory's research cluster which is comprised of the following components:

- 2 *Class A* nodes - Two 10 core, 20 thread Intel Xeon E5-2650 v3 @ 2.30GHz and 128 GB RAM

Test suites	**Description**	**Type**
Audio Encoding	Runtime measurement to encode WAV file to different audio formats.	CPU
Cryptography	Cryptography tests such as OpenSSL and GnuPG.	CPU
Network Loopback	Computer's networking performance testing.	Network
Avrora	Multithreaded AVR microcontrollers simulator.	CPU
Batik	Produces Scalable Vector Graphics images.	Memory
Eclipse	Non-GUI jdt performance tests for the Eclipse IDE.	CPU
Jython	Interprets the pybench Python benchmark.	CPU
Pmd	Multithreaded Java source code analysis.	CPU
Tradebeans	Daytrader benchmark run on GERONIMO with an in-memory H2 DB.	Memory
H2	Executes transactions against a model of a banking application.	Memory
Xalan	Multithreaded XML to HTML converter.	Mixed
Sunflow	Renders a set of images using ray tracing.	CPU
miniFE[49]	Finite element generation, assembly and solution for an unstructured grid problem.	CPU
DGEMM[62]	Multi-threaded, dense-matrix multiplication.	CPU
STREAM[31]	Calculates sustainable memory bandwidth and the computation rate for simple vector kernels.	Memory

Table 4.4. *Workload benchmarks*

- 2 *Class B* nodes - Two 8 core, 16 thread Intel Xeon E5-2640 v3 @ 2.60GHz and 64 GB RAM
- 4 *Class C* nodes - Two 6 core, 12 thread Intel Xeon E5-2620 v3 @ 2.40GHz and 64 GB RAM

The classification of the nodes into power classes in the Stratos cluster was done based on the potential power consumption by processors on each of those nodes. Class A nodes have the highest Thermal Design Power (TDP), followed by Class B, and finally Class C. The higher the TDP, the more Watts a processor is expected to consume as per the chip manufacturer specifications. Each node runs 64-bit Linux 4.4.0-45 and shares an NFS server. Apache Mesos 1.0.1 is deployed as the cluster manager. For some experiments, Apache Aurora 0.16.0 is used as the sole framework to schedule workloads. For other experiments, our Electron framework is used. Docker 1.12.3 is used as the container technology. Performance Co-Pilot [55] is used to collect metrics for all nodes in the cluster. These metrics include energy measurements from RAPL counters and various statistics about CPU and memory usage from each worker node's Linux kernel. An Ansible playbook is used to statically cap all nodes in the cluster using the Linux Powercapping framework[58] for our static capping experiments. For our experiments with Extrema dynamic capping, the Electron framework is able to interact directly with the hosts to set a power cap. All benchmarks are run inside Docker containers to maintain workload environment consistency.

Each DaCapo workload possesses distinct characteristics[6], as listed in Table 4.4. The workloads with which we assess performance were derived from the DaCapo Benchmark suite, Phoronix Benchmark suite, MiniFE from Mantevo [49], as well as Stream and Dgemm from NERSC [53]. The DaCapo benchmark `tradebeans` simulates a cloud workload with memory and network intensive components. The remaining benchmarks simulate diverse types of workloads varying from highly parallel workloads to highly serial tasks.

4.5 Performance Analysis

	Aurora	Electron FF	Electron BP
Makespan (s)	1441	1532	1640
Power Max (W)	1066.52	1231.38	1209.15
Power Mean (W)	729.28	684.87	602.09
Power Median (W)	788.87	728.30	633.81
90th Percentile (W)	955.55	924.40	684.45
95th Percentile (W)	981.73	970.09	707.96

Table 4.1: Power consumption statistics for Aurora, Electron FF, and Electron BP.

Aurora vs. Electron First Fit (Electron FF)

In Figures 4.2 to 4.4, we can observe that Aurora and Electron with a First Fit policy exhibit similar performance. The similarity in performance is backed by a 1% difference in the 95^{th} percentile for power consumption. The difference in the mean power consumption, however, is lower by 6% for Electron FF as compared to Aurora's mean power consumption. A notable difference between the two power profiles occurs at time 859 where Aurora continues to fluctuate between 800 and 1000 Watts while Electron FF begins to decrease power consumption, experiencing max peaks of 710 Watts at time 914 as can be seen in Figure 4.3. While the energy consumption is virtually the same for both Electron FF and Aurora with only a 2 kilojoule difference, there is a decrease of 6% in the mean CPU Time used to finish all computations by

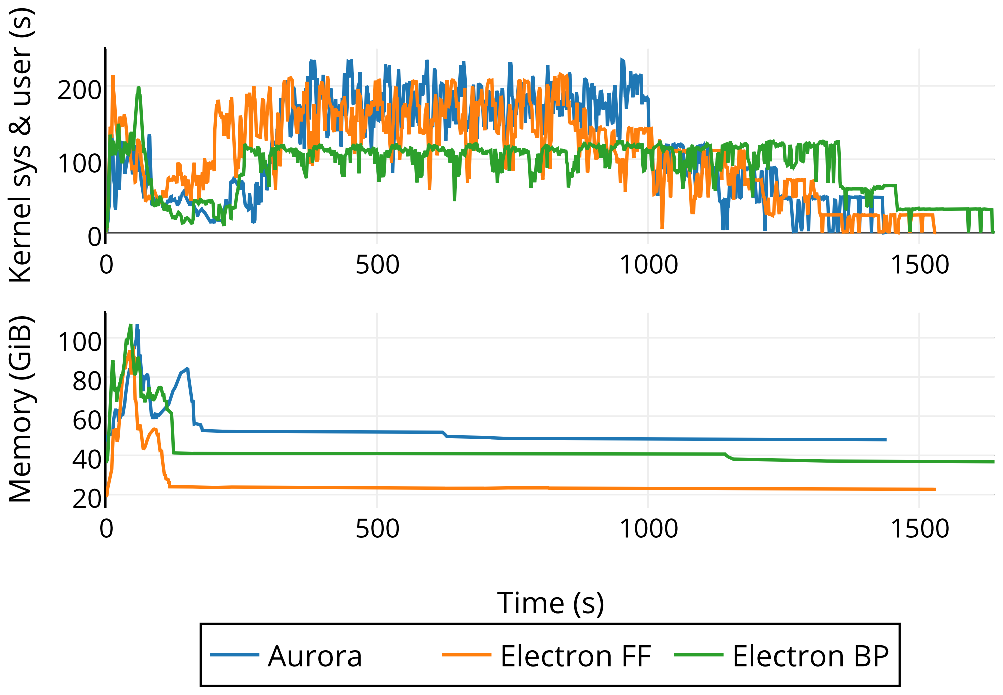

Figure 4.2: Cluster-wide CPU & Memory Utilization of Aurora and Electron without a power cap.

Electron FF. Finally, the difference in memory consumption between Electron FF and Aurora is large, with Electron FF experiencing a mean memory utilization that is 51% lower. We speculate that this difference in memory usage may be due to inactive memory left behind by earlier experiments making the memory usage for Electron FF an outlier. Further investigation needs to be done to understand the large difference in memory utilization observed in Figure 4.2 between Aurora and Electron FF.

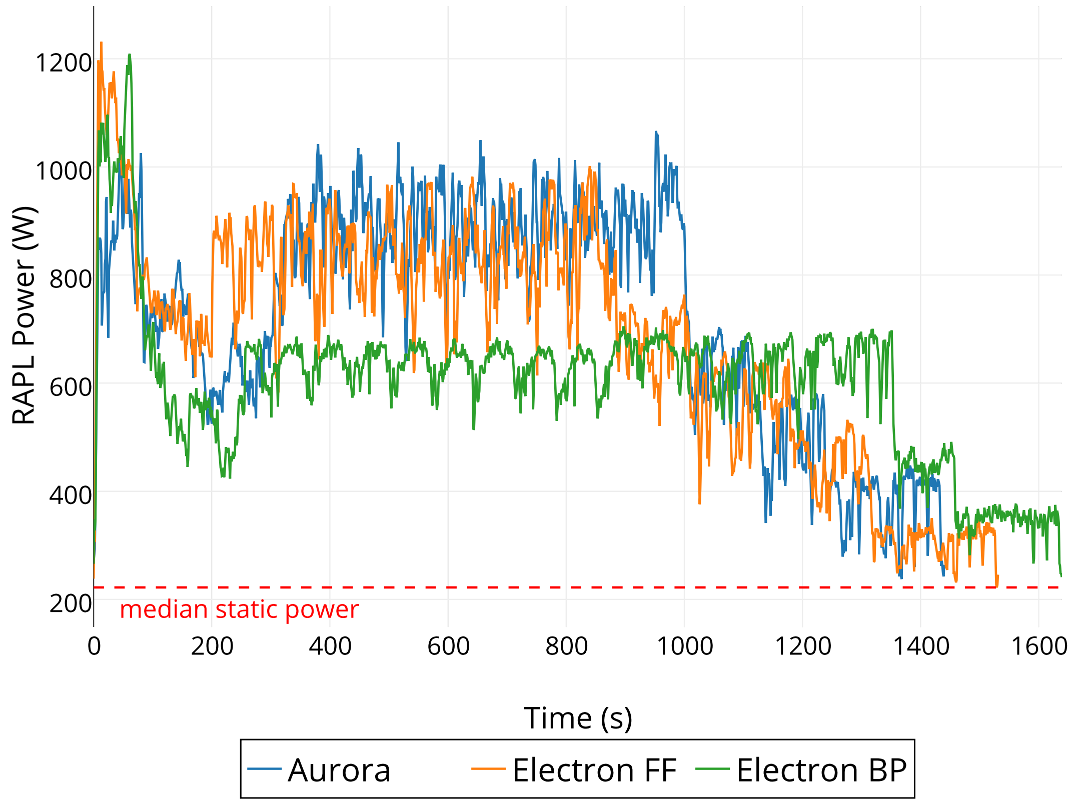

Figure 4.3: Cluster-wide Power Consumption of Aurora and Electron without a power cap.

Electron FF vs. Electron Bin Packing (Electron BP)

A comparison between Electron FF and Electron Bin Packing can also be seen in Figures 4.2 to 4.4. Electron BP suffers an increase in makespan compared to Electron FF of 7% and 14% compared to Aurora. However, the 95th percentile of power consumption for the Electron BP run is decreased by 27% compared to Electron FF and a similar percentage when compared to Aurora. Overall energy consumption is decreased by 6% compared to Electron FF and decreased by 6% compared to Aurora.

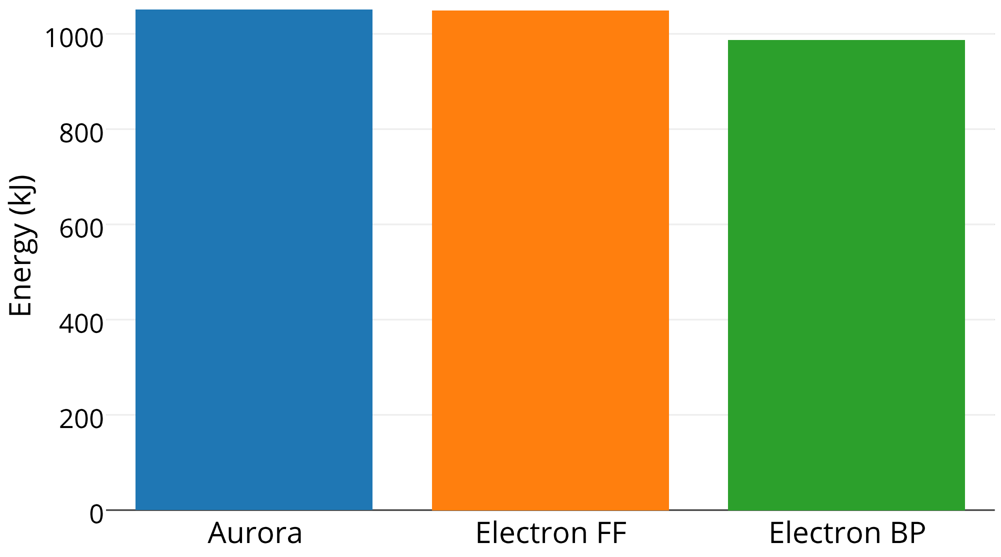

Figure 4.4: Cluster-wide Energy Consumption of Aurora and Electron without a power cap.

The mean CPU Time needed to finish all workloads by Electron BP is 19% lower than Electron FF and 23% lower than Aurora. Mean memory consumption is 19% lower compared to Aurora.

	Aurora-S	Electron FF-S	Electron BP-S
Makespan (s)	1643	1570	1616
Power Max (W)	886.14	929.28	948.05
Power Mean (W)	611.87	634.20	532.34
Power Median (W)	700.02	706.04	546.42
90th Percentile (W)	816.56	820.59	575.15
95th Percentile (W)	827.75	833.44	735.26

Table 4.2: Power consumption statistics for Aurora, Electron FF, and Electron BP under a static power cap. Electron BP experiences the highest power peaks and makespan but the lowest mean, median, 90th percentile, and 95th percentile.

4.5.1 Effects of Static Power Capping

Aurora-S

Setting a static cap for the entire cluster resulted in many improvements when compared to the uncapped run. Aurora-S experienced a 16% reduction in the 95th percentile of power across the cluster. The max peak power was reduced by 17% while overall energy consumption was reduced by 4%. However, static capping also resulted in a 14% increase in makespan. Mean CPU time decreased by 3% while memory utilization increased by 7%.

Electron FF-S

Electron FF run under static capping (Electron FF-S) experienced a 14% reduction in the 95th percentile of power and a 25% reduction of the max peak power draw when compared to its uncapped run. Electron FF-S experienced a makespan increase of 2%. Overall energy was reduced by 5%. Mean CPU time increased by 12%. The increase in CPU time can be attributed to the fact that the clock speed of the processor is decreased by RAPL in order to satisfy the power cap placed. Thus, the number of

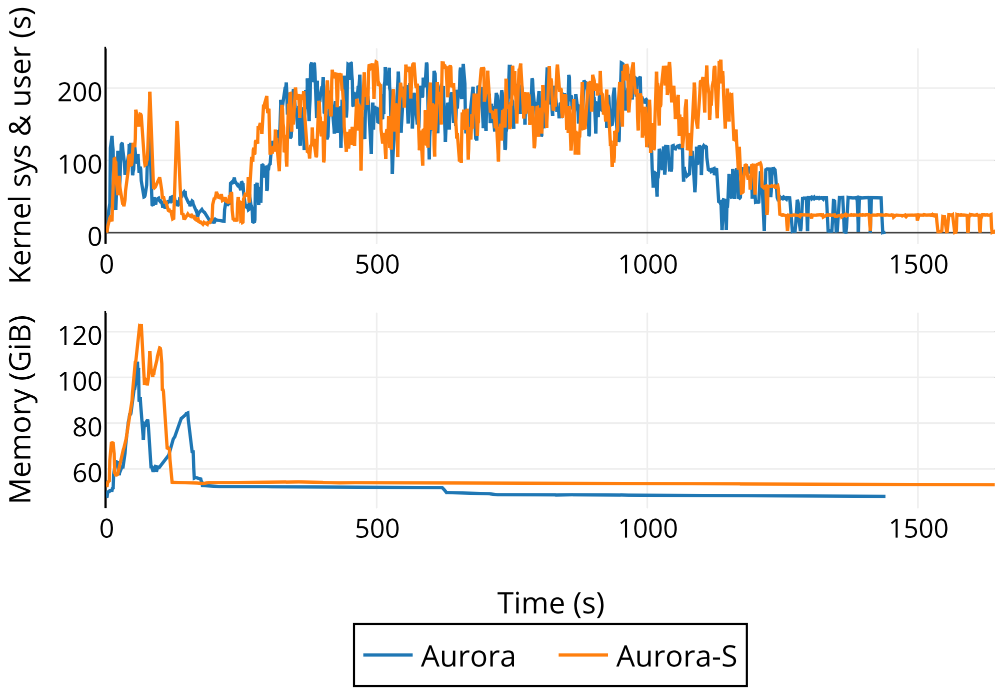

Figure 4.5: Cluster-wide CPU & Memory Utilization for Aurora and Aurora running under a static power cap.

clock ticks taking place in a second is reduced, and so is the amount of work a core is able to do. Thus, it takes more CPU time to do the same work that was done by the uncapped run.

Electron BP-S

Electron BP-S finished all workloads with an increase in makespan of 8% in comparison to the uncapped run. The power in the 95th percentile increases as well by 4%. However, the max peak power experienced a decrease of 22% and overall energy was reduced by 5%. Mean CPU Time increased 9% while mean memory usage increased

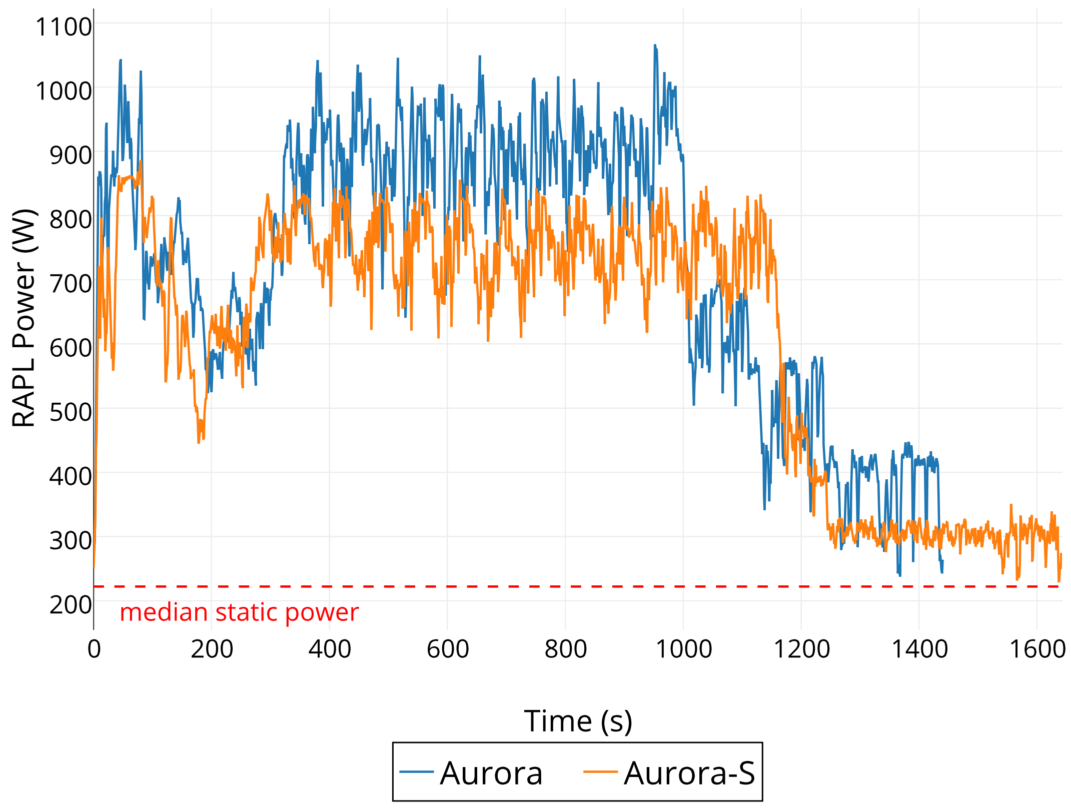

Figure 4.6: Cluster-wide Power Consumption for Aurora and Aurora running under a static power cap.

by 24%.

4.5.2 Effects of Extrema Power Capping

Electron FF-E

Electron with a First Fit policy running under Extrema capping (Electron FF-E) experiences a similar makespan to the uncapped run, differing by less than 1%. However, when compared to the static capped run, Electron FF-E experiences a decrease in

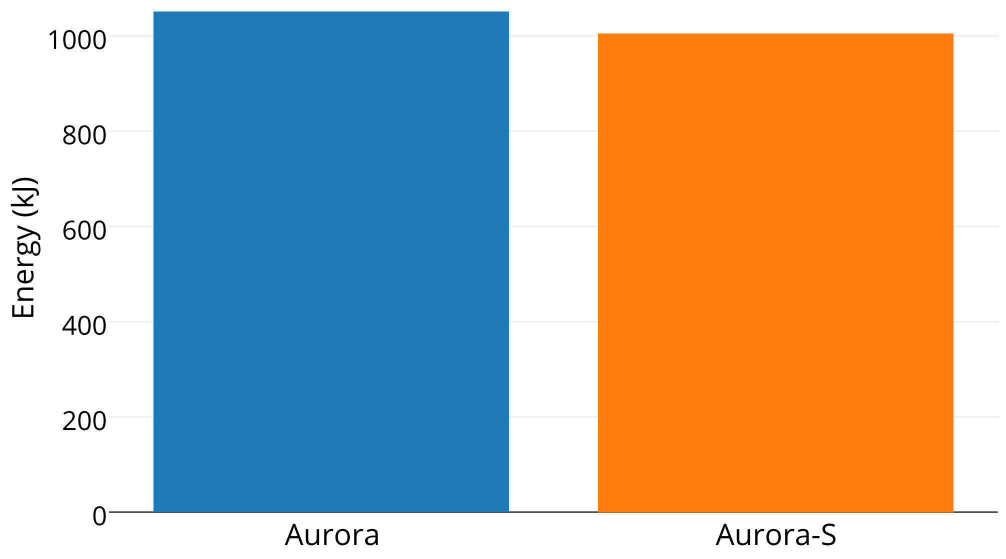

Figure 4.7: Cluster-wide Energy Consumption for Aurora and Aurora running under a static power cap.

	Electron FF-E	Electron BP-E
Makespan (s)	1533	1616
Power Max (W)	1138.62	1000.31
Power Mean (W)	627.75	525.81
Power Median (W)	655.40	524.48
90th Percentile (W)	762.83	575.28
95th Percentile (W)	786.04	724.46

Table 4.3: Power consumption statistics for Electron FF and Electron BP under an extrema dynamic power cap. The only metric in which Electron FF bests the performance of Electron BP is on makspan.

makespan of 2%. The power readings in the 95^{th} percentile are improved in Electron FF-E by 19% and 6% when compared to Electron FF and Electron FF-S respectively.

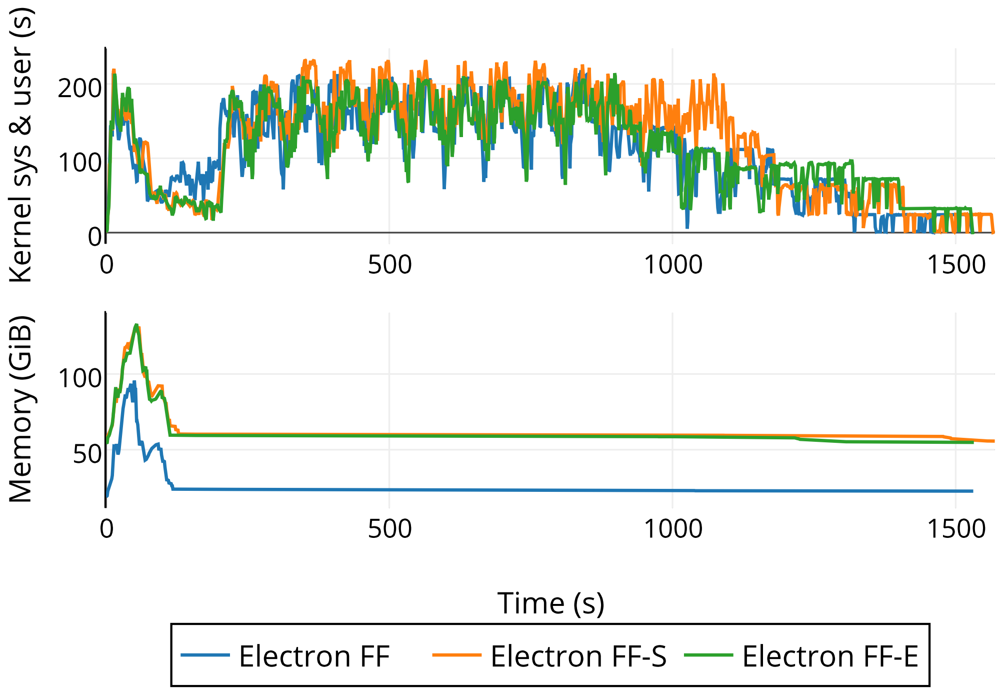

Figure 4.8: Cluster-wide CPU & Memory Utilization for Electron First Fit, Electron First Fit running under static cap, and Electron First Fit running under the extrema dynamic cap.

The max peak power of Electron FF-E decreased by 8% compared to the uncapped run while it fared worse against the static capped run with an increase of 23%. The difference in the max peak power draw between Electron FF-S and Electron FF-E is the result of the Extrema capping algorithm being reactive to power trends. As such, it's only after the power spike which occurs when all benchmarks are scheduled across the cluster, that it reacts to the trend by capping nodes. Future work will explore adding the ability to mitigate these initial power spikes. Energy consumption is lowered by 8% compared to Electron FF and 3% to Electron FF-S. Mean CPU time

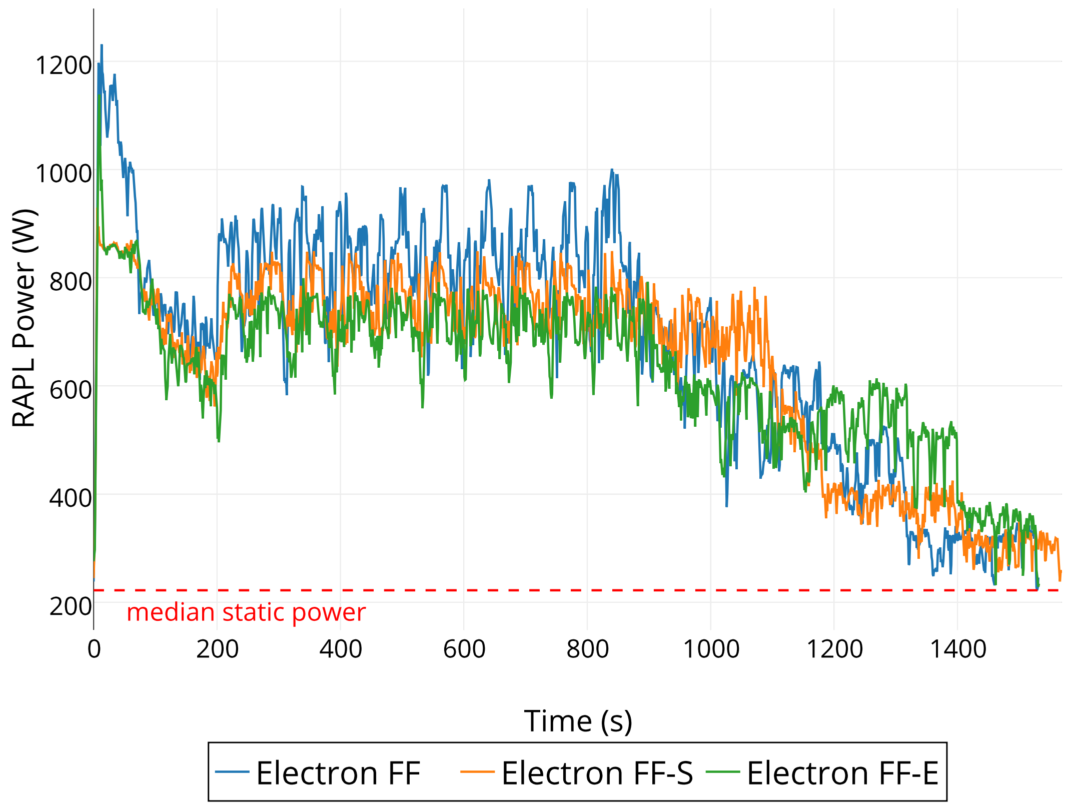

Figure 4.9: Cluster-wide Power Consumption for Electron First Fit, Electron First Fit running under static cap, and Electron First Fit running under the extrema dynamic cap.

is 4% higher compared to the uncapped run and 8% lower compared to the statically capped run. Mean memory consumption increased by 137% compared to Electron FF but was only 2% lower than Electron FF-S, leading us to believe that the memory consumption experienced by Electron FF is an outlier.

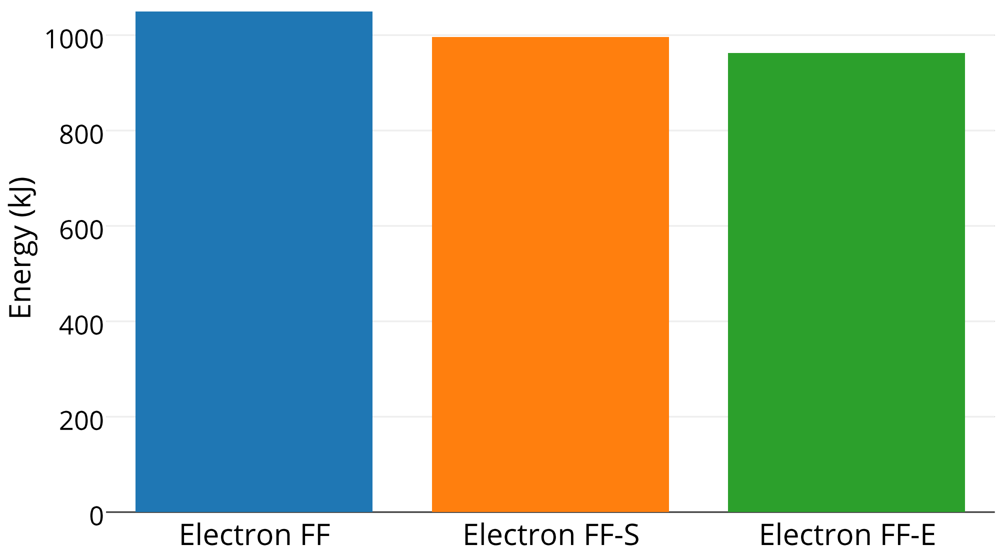

Figure 4.10: Cluster-wide Energy Consumption for Electron First Fit, Electron First Fit running under static cap, and Electron First Fit running under the extrema dynamic cap.

Electron BP-E

Electron with a Bin Packing policy running under Extrema capping (Electron BP-E) finishes at roughly the same time as Electron BP with a 2% difference in makespan between Electron BP and Electron BP-E. The 95^{th} percentile of power consumption for Electron BP-E is higher by 2% than Electron BP and lower by 2% when compared Electron BP-S. The max peak power consumption is decreased compared to the uncapped run by 17%, while it resulted in an increase of 6% compared to the static capped run. These results in max peak power are also the result of initial power

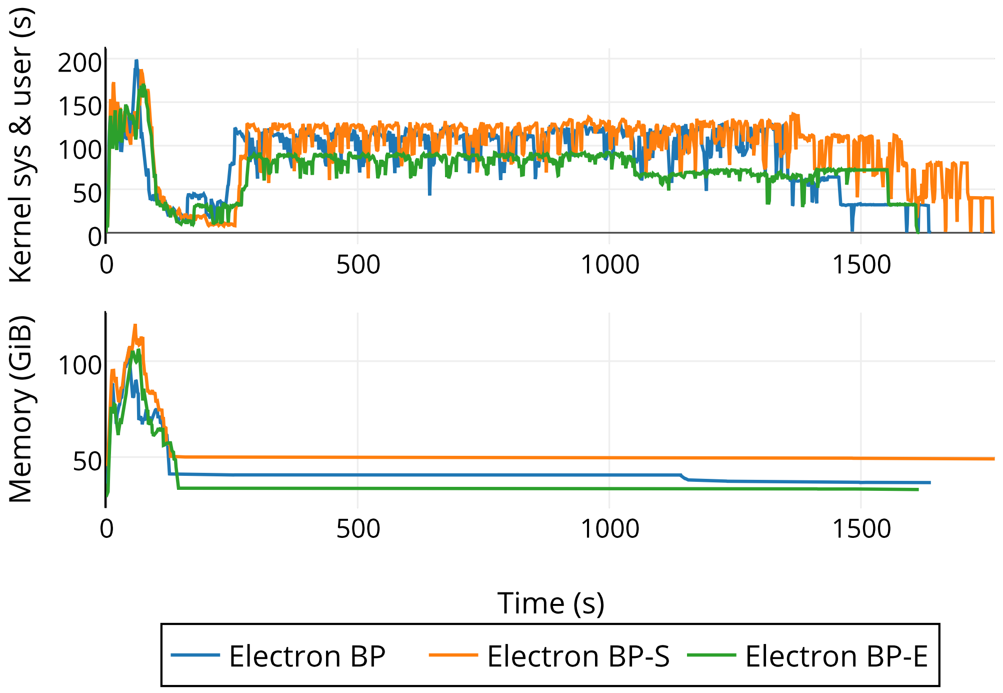

Figure 4.11: Cluster-wide CPU & Memory Utilization of uncapped Electron Bin Packed, statically capped Electron Bin Packed and Electron Bin Packed running under Extrema capping.

peaks when jobs are started as described in section 4.5.2.

4.6 Conclusion

Static capping is not particularly suited for the co-locating nature of cloud workloads nor the abstraction layer provided by Mesos as it results in larger makespans. Dynamic capping, such as Extrema, is a step in the right direction as it is able to apply power capping without having the negative impact that static capping has on

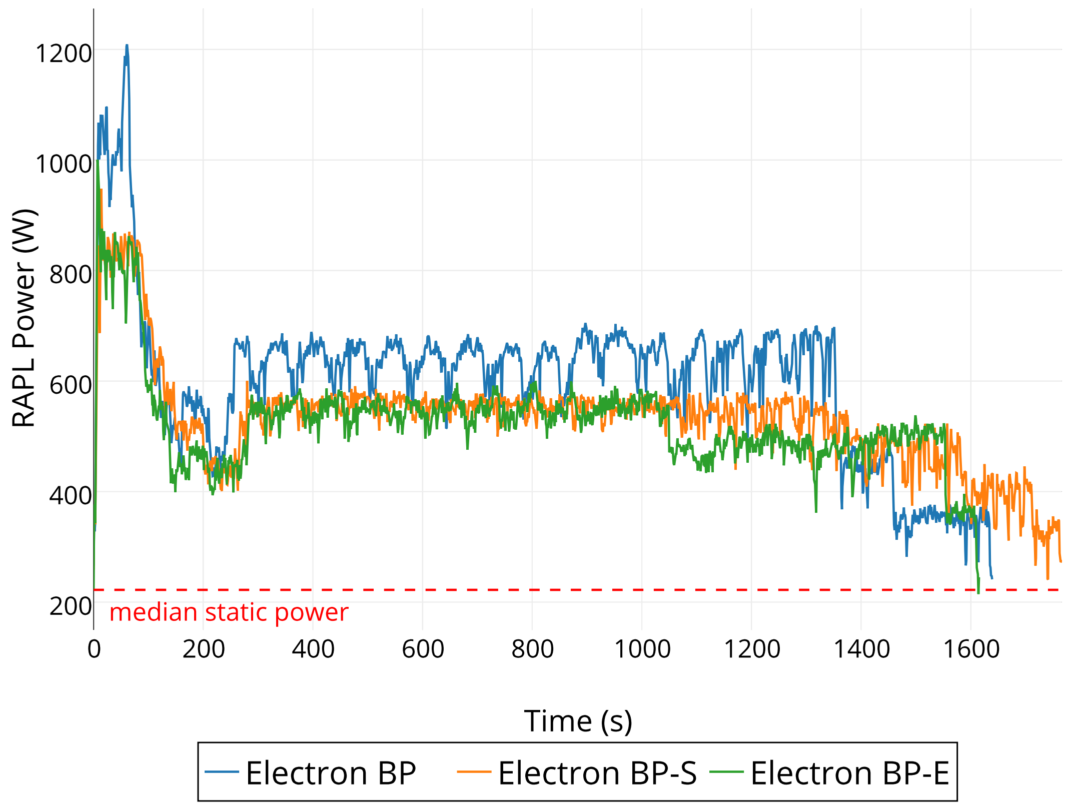

Figure 4.12: Cluster-wide Power Consumption of uncapped Electron Bin Packed, statically capped Electron Bin Packed and Electron Bin Packed running under Extrema capping.

makespan. However, Extrema is less effective in reducing max peak power in comparison to static capping due to its reactive nature. The ideal dynamic capping solution should be able to accurately predict the power consumption of co-located workloads and power cap accordingly. There are several variables when it comes to making such a prediction – Do the peaks of these benchmarks align when started at the same time? How does the start time of different benchmarks affect the power consumption of their co-location? We will explore such ideas in future work.

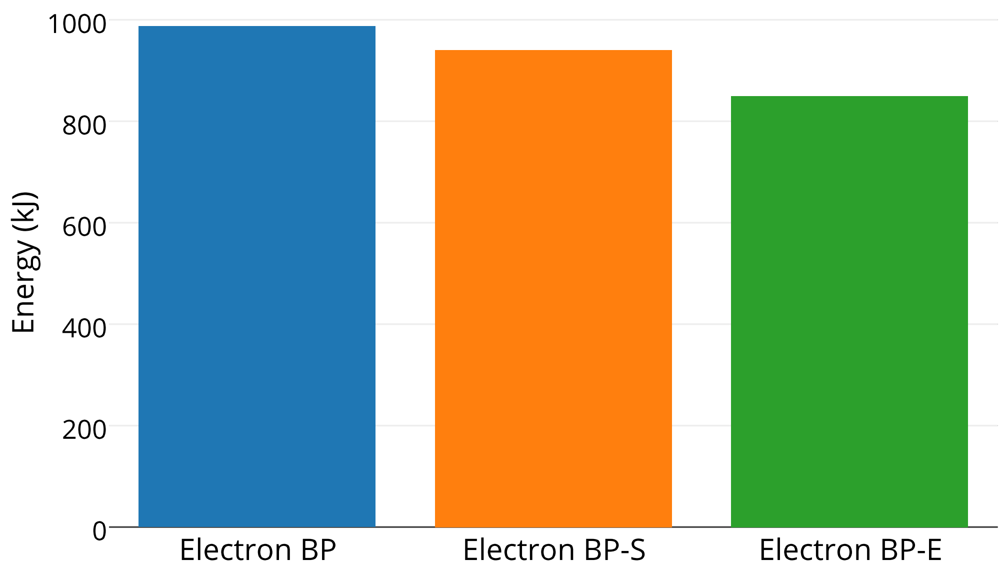

Figure 4.13: Cluster-wide Energy Consumption of uncapped Electron Bin Packed, statically capped Electron Bin Packed and Electron Bin Packed running under Extrema capping.

Our framework, Electron, which is available from our laboratory's Bitbucket repository, has the ability to be configured for scheduling policies and power capping strategies. It is designed to help data center managers make informed decisions on power spikes and energy consumption when Mesos is used as the infrastructure-wide resource manager.

We list our findings from our experiments which also serve as an example of the insights that our experimental setup can provide for other workloads.

- When compared to results from Electron's First Fit, Bin Packing manages to maintain lower power peaks. It also results in lower CPU time. This is caused by an increase in clock speed when nodes are bin packed, resulting in more work done per second.

- Compared to Aurora, Electron's Bin Packing approach reduces the total energy usage by 6% for the tested benchmarks. Additionally, the median power usage is reduced by 20%.
- Static capping works well when the power consumption of the benchmarks in the workload stays below or slightly above the static cap value.
- Compared to the uncapped run, the statically capped runs show a reduced power ceiling ranging from a decrease of 17% for Aurora to a decrease of 25% for Electron FF.
- When comparing all results against Aurora's performance as a baseline, Electron BP under Extrema experiences the largest improvements in power and energy consumption with 19% reduction in energy consumption and 28% reduction in mean power consumption.

Chapter 5

Exploiting Efficiency Opportunities Based on Workloads with Electron on Heterogeneous Clusters

Resource Management tools for large-scale clusters and data centers typically schedule resources based on task requirements specified in terms of processor, memory, and disk space. As these systems scale, two non-traditional resources also emerge as limiting factors: power and energy. Maintaining a low power envelope is especially important during *Coincidence Peak*, a window of time where power may cost up to 200 times the base rate. Using Electron, our power-aware framework that leverages Apache Mesos as a resource broker, we quantify the impact of four scheduling policies on three workloads of varying power intensity. We also quantify the impact of two dynamic power capping strategies on power consumption, energy consumption, and makespan when used in combination with scheduling policies across workloads. Our experiments show that choosing the right combination of scheduling and power capping policies can lead to a 16% reduction of energy and a 37% reduction in the 99th percentile of power consumption while having a negligible impact on makespan and resource utilization.

5.1 Introduction

Resource management in large heterogeneous clusters is essential both to effectively use the available resources (such as processors, memory, and storage) and to reduce the cost in terms of the power envelope and energy usage. Apache Mesos [25] has emerged as the leader in the resource management space in the open source community. Mesos is akin to a distributed operating system, pooling together the available cores, system memory, and disk space for consumption by the applications on the cluster. Mesos' two-level scheduling scheme, along with its fair resource distribution policy, has shown to be successful for massive workload execution. Other efforts, such as Hadoop's YARN [71], Docker Swarm [51], and Kubernetes [8], work in a similar manner. These cluster management tools have generated interest in the science and academic computing environments due to the recent availability of virtualized clouds for production use such as JetStream [67] and Chameleon [9]. However, Mesos, YARN, Swarm, and Kubernetes do not have support for considering energy budgets and power envelope in their off-the-shelf packages.

Mesos, Hadoop's YARN, Docker's Swarm, Google's Kubernetes and other tools in this space, are designed to allow co-scheduling of workloads on worker nodes. As tasks are co-scheduled, the power they draw from the node is dependent on how efficiently co-scheduled tasks use hardware resources and how the peak power draws of tasks align. *Coincident Power* is the total power drawn from the cluster at any time. This value is dependent on the power consumed by all the tasks executing in the same instant. This includes the supporting software stack and hardware components. Ensuring that the cluster's desired power envelope is not breached requires workload shifting to ensure that the power peaks of various tasks do not align. Maintaining low power usage is especially important during the *Coincidence Peak*, a window of time where power may cost up to several times the base rate [42].

It is known that optimally co-scheduling applications to minimize peak power usage can be reduced to a multi-dimensional Bin-Packing problem and is NP-Hard [46]. Previously [16], we used Mesos and Aurora [4] to demonstrate how a policy driven approach, involving Bin-Packing workloads, can effectively reduce peak power consumption and energy usage. In our previous work [14], we introduced a pluggable power aware framework for Mesos, Electron. In this work, we deploy Electron with three different workloads, four different scheduling algorithms, and two different power capping strategies to quantify the effects that the different combinations of these three components have on power consumption, total energy consumption, and makespan.

Our workloads are composed of the DaCapo benchmarks [6], Phoronix benchmarks [57], and Scientific micro-benchmarks. The DaCapo benchmark suite is a set of open source, real-world applications that exercise the various resources within a compute node, while the Phoronix and scientific workloads are microbenchmarks. Our approach measures and monitors the power usage of CPU and Memory for each node using fine-grained power profiles provided by Intel's Running Average Power Limit (RAPL) [10] counters via the Linux Powercapping framework [58]. We use the power profiling data for a given benchmark and determine the approximate power usage.

We make the following contributions in this paper:

- We profile several different, well understood benchmarks and classify them using *k-means* clustering into low power consuming tasks and high power consuming tasks based on their power consumption.

- In contrast to the single type of workload used in our previous work [14], we use the benchmark classification to construct three kinds of workloads, each varying in power consumption. The different workloads are then used to quantify the power, energy, and makespan characteristics of the combinations of various scheduling policies and power capping strategies.

- We include two new scheduling policies and analyze their effect on power consumption, energy consumption, and makespan when used to schedule the different categories of workloads.

- We introduce a new power capping strategy to overcome certain limitations of the power capping strategies discussed in our previous work [14] and to further dampen large fluctuations in power consumption.

- We make recommendations based on our findings, on how different scheduling policies and power capping strategies should be used to satisfy Coincident Peak constraints and energy consumption requirements for a Mesos-based cluster.

5.2 Electron

In our previous work [14], we introduced Electron, a pluggable power conscious framework that runs on Mesos. A high level view of Electron's architecture is shown in

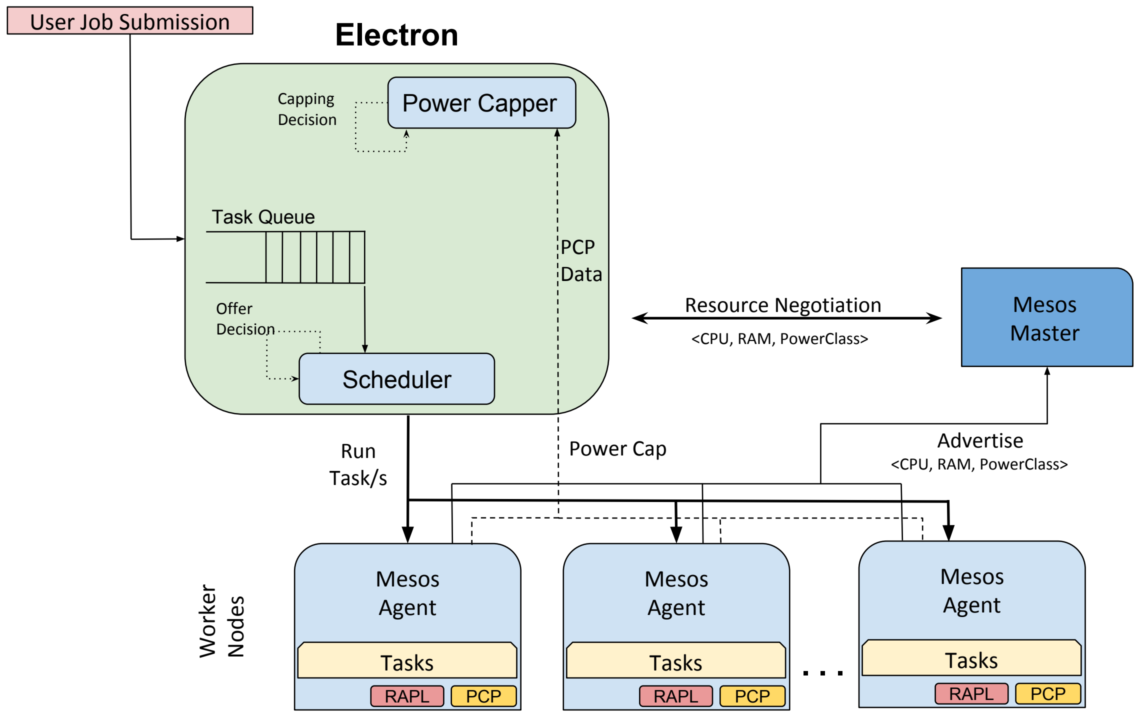

Figure 5.1: Architecture of Electron and Mesos. Electron's Scheduler component receives resource offers from Mesos and uses them to schedule tasks on worker nodes. The Power Capper component analyzes the power consumption of the worker nodes and decides whether or not to power cap one or more worker nodes.

Figure 5.1. Electron was built with both pluggable scheduling policies and pluggable power capping strategies. Electron is comprised of three main components: Task Queue, Scheduler, and Power Capper.

Task Queue: Maintains the tasks that are yet to be scheduled. **Scheduler**: Checks whether the resource requirements for one or more tasks, in the Task Queue, can be satisfied by the resources available in the Mesos resource offers. If yes, those tasks are scheduled on the nodes corresponding to the consumed resource offer.

Power Capper: Responsible for power capping one or more nodes in the cluster, through the use of RAPL [61]. The Power Capper monitors the power consumption of the nodes in the cluster, which is retrieved through the use of Performance Co-Pilot [55]. A power capping policy that is plugged into Electron uses this information to make the decision to power cap one or more nodes in the cluster.

5.2.1 Power Classes

We categorized the machines in our cluster into four power classes: A, B, C, and D, based on their Thermal Design Power (TDP). We made each node advertise its power class to the Mesos master, leveraging the feature of Mesos to allow agents to advertise arbitrary resources. The specifications of the machines belonging to each power class is described in Section 5.3.1.

5.2.2 Consuming Mesos Offers

First-Fit (FF)

For each offer the framework receives, it finds the first task in the job queue whose resource constraints are satisfied by the resources available in the offer. If a match is made between an offer and a task, the offer is consumed in order to schedule the matching task. Otherwise, it moves on to a new resource offer and the process of finding a suitable task is repeated.

Bin-Packing (BP)

For each offer that is received by the framework, tasks are matched from the priority queue keyed by an approximation of the worst case power consumption using Median of Medians Max Power Usage (M^{3}PU) described in more detail in Section 5.3.3. If a task's resource requirements are not satisfied by the remaining resources, the next task in the queue is evaluated for fitness. We repeat this process until no task from the queue fits in the remaining resources. The offer is then consumed and the set of tasks evaluated to fit are scheduled to run.

The distribution of the workload when using Bin-Packing as the method of consuming Mesos offers for a Moderate Power Consuming workload is shown in Figure 5.2. Bin-Packing can lead to uneven distribution of tasks such that one class of machines handles roughly 44% of the work, compared to the other three power classes.

Max-Min (MM)

Although Bin-Packing reduces peak power consumption compared to a First-Fit policy, BP commonly leads to excessive co-location of high power consuming tasks which creates contention for resources, which results in stragglers and therefore a larger makespan. Max-Min is able to address this issue by picking a mixed set of tasks from the queue. Max-Min uses a double-ended queue *(deque)* sorted in non-decreasing M^3PU values, alternating between attempting to schedule tasks from the front and the back of the *deque*. If a task fits in the offer, the count of available resources is reduced and the process is repeated. If there are no more tasks that fit in an offer from the *deque*, Max-Min moves on to the next offer.

The distribution of tasks when Max-Min is used as the scheduling policy for a Moderate Power Consuming workload is shown in Figure 5.3. Max-Min results in the resources contained in an offer to be consumed in quicker succession, thereby leading to a better distribution of the workload across the cluster. Notice in the figure that the workload is better distributed among the power classes as compared to Figure

5.2. However, *Max-Min* does not show a noticeable improvement in the distribution of high power consuming tasks when compared to *Bin-Packing*.

Max-GreedyMins (MGM)

Through experimentation we found that Max-Min has a significant impact on makespan when there is a higher proportion of low power consuming tasks. We created *Max-GreedyMins (MGM)* to counter MM's impact on makespan, and to further reduce peak power and energy consumption. MGM consumes offers by packing the low power consuming tasks at a faster rate than the high power consuming tasks. Like MM, unscheduled tasks are stored using a *deque* sorted in non-decreasing M^{3}PU. *MGM*, as shown in Algorithm 3, attempts to pack tasks into an offer by picking one task from the end of the *deque* and as many tasks from the beginning of the *deque* until no more tasks can be fit into the offer. Once no more tasks fit in the resources available from the offer, the policy moves on to the next Mesos offer and repeats the process.

The distribution of the workload when *MGM* is used as the scheduling policy for a Moderate Power Consuming workload is shown in Figure 5.4. Not only does *MGM* more evenly distribute the workload across the cluster, but it also betters the distribution of high power consuming tasks when compared against Figure 5.3. This increase in distribution of high power consuming tasks would help reduce starvation, thereby also reducing the impact on makespan.

Algorithm 3 Max-GreedyMins

```
 1: $sortedTasks$ – Tasks to schedule sorted in non-decreasing order by their corre-
    sponding M$^3$PU value.
 2: $offer$ – Mesos offer.
 3: procedure Max-GreedyMins($offer$, $sortedTasks$)
 4:     for task in $sortedTasks$.Reverse() do
 5:         if Fit($offer$.UnusedResources(), task) then
                $offer$.schedule(task)
                $sortedTasks$.remove(task)
                break
 6:         end if
 7:     end for
 8:     for task in $sortedTasks$ do
 9:         for instance in task.Instances() do
10:             if Fit($offer$.UnusedResources(), task) then
                    $offer$.schedule(task)
                $sortedTasks$.remove(task)
11:             else
                    break
12:             end if
13:         end for
14:     end for
15: end procedure
```

5.2.3 Power Capping Policies

Extrema

In our previous work, we presented [14] a dynamic power capping strategy, Extrema, which is able to make trade-offs between makespan and power consumption. Extrema reacts to power trends in the cluster and restraints the power consumption of the cluster to a power envelope defined by a high and low threshold. If the cluster's Average Historical Power (AHP) exceeds the high threshold, the node consuming the highest power is power capped to half its Thermal Design Power (TDP). TDP is the maximum expected power (turned to heat) that is expected to be dissipated, as listed

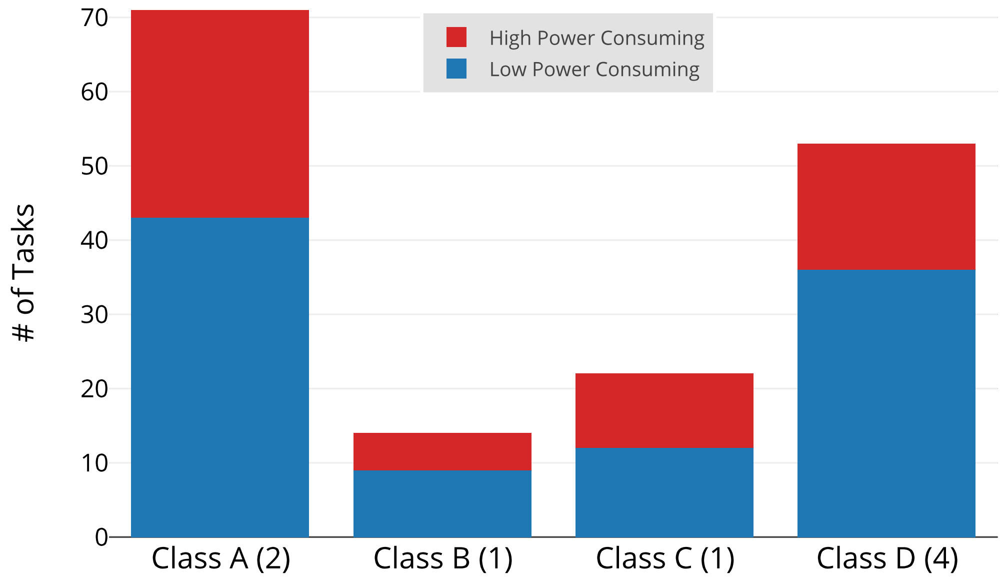

Figure 5.2: Distribution of tasks by classification for a Moderate Power Consuming workload when using Bin-Packing. The number of nodes for each power class is shown in parentheses. This figure shows Class A nodes processing more power intensive tasks than Class D nodes despite having half as many workers when using the Bin-Packing strategy.

by the chip manufacturer. On the other hand, if the cluster's AHP is lower than the low threshold, a node is uncapped. Nodes are fully uncapped in the reverse order in which they were capped. Extrema is successful in maintaining the power profile within a defined envelope, reducing power peaks while having a subdued impact on makespan.

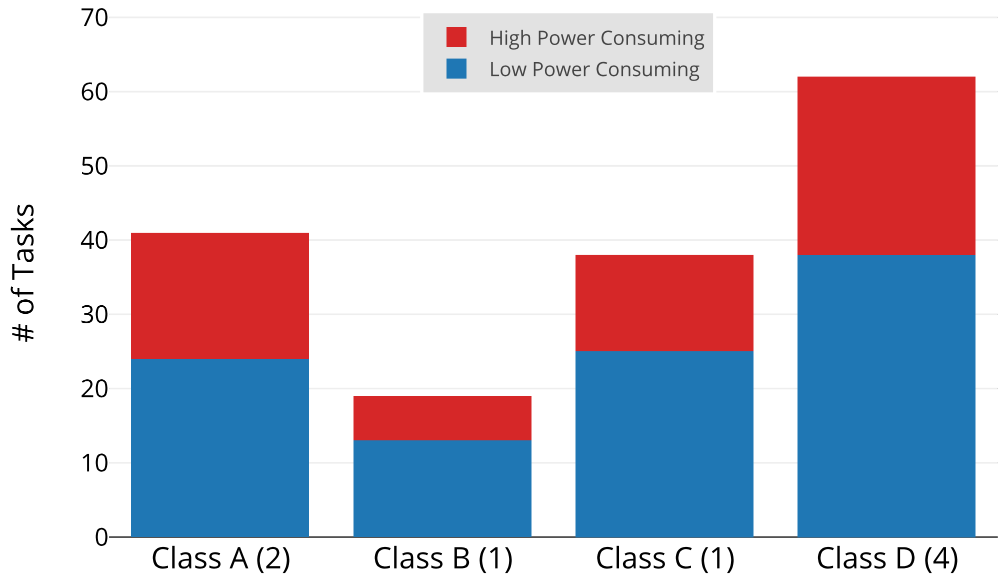

Figure 5.3: Distribution of tasks by classification for a Moderate Power Consuming workload when using Max-Min. The number of nodes for each power class is shown in parentheses. This figure shows Class D bearing a larger burden than the rest of the classes with Max-Min as the scheduling strategy.

Progressive Extrema

Through experimentation we have identified a few limitations exhibited by Extrema:

1. If every node in the cluster has already been capped but the AHP is still above the High threshold, Extrema is unable to perform any new action that may bring the AHP down.
2. As Extrema caps nodes to 50% of their TDP and uncaps them to their full TDP, large differences in these values can result in nodes experiencing high

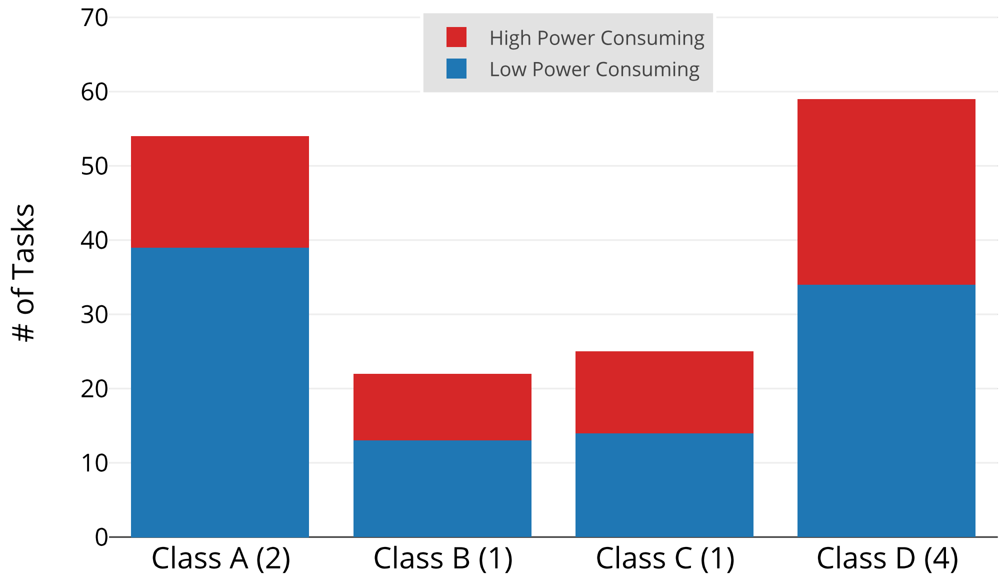

Figure 5.4: Distribution of tasks by classification for a Moderate Power Consuming workload when using Max-GreedyMins. The number of nodes for each power class is shown in parentheses. This figure shows Class A and Class D completing about the same number of tasks in number with Class D processing more power intensive tasks when using a Max-GreedyMins strategy to schedule workloads.

power fluctuations.

3. Extrema requires high and low thresholds to be manually predefined. It follows that prior knowledge of the workload greatly benefits the configuration of the high and low thresholds and by extension, the efficacy of Extrema.

While the last drawback still remains an open problem, we address the first two drawbacks through a modified version of Extrema named *Progressive Extrema*. Progressive

Extrema, described in Algorithm 4, is similar to Extrema except for one key difference: power capping is applied in phases. Whereas picking a previously capped node as a victim in Extrema resulted in a *no-op*, in Progressive Extrema the same scenario results in a harsher capping value for the victim. In the initial phase of Progressive Extrema's design, capping history is maintained for each node in an in-memory data store. When a victim node is chosen for uncapping the previous cap value in the node's history is used. Since these values are determined algorithmically, this uses O(n) additional memory. It also uses a *Cap Limit* that defines the floor value beyond which a node should not be capped.

5.3 Experiments

5.3.1 Setup

Our experiments were conducted on a research cluster which comprises the following components:

- 2 *Class A* nodes - Two 10 core, 20 thread Intel Xeon E5-2650 v3 @ 2.30GHz and 128 GB RAM per node
- 1 *Class B* node - Two 8 core, 16 thread Intel Xeon E5-2640 v3 @ 2.60GHz and 128 GB RAM per node
- 1 *Class C* node - Two 8 core, 16 thread Intel Xeon E5-2640 v3 @ 2.60GHz and 64 GB RAM per node
- 4 *Class D* nodes - Two 6 core, 12 thread Intel Xeon E5-2620 v3 @ 2.40GHz and 64 GB RAM per node

Algorithm 4 Progressive Extrema Capping

```
procedure ProgExtrema($Threshold$, $InitCap$, $CapLimit$)
    $ClusterAvg \leftarrow Avg_{Running}(ClusterPower)$
    if ClusterAvg > Threshold.Hi then
        $Victims \leftarrow Sort_{non-inc}(AvgPowerNode[...])$
        $uncappedVictimFound \leftarrow false$
        for victim in Victims do
            if victim not in $CappedVictims$ then
                Cap(victim, InitCap)
                $CappedVictims[victim.Host] \leftarrow InitCap$
                $uncappedVictimFound \leftarrow true$
            end if
        end for
        if uncappedVictimFound == false then
            for victim in $CappedVictims$ do
                if victim.curCap > $CapLimit$ then
                    $newCap \leftarrow$ victim.curCap $\div 2$
                    $Cap(victim, newCap)$
                    $CapVictims[victim.Host] \leftarrow newCap$
                end if
            end for
        end if
    end if
    if ClusterAvg < Threshold.Low then
        $victim \leftarrow MaxCapped(CappedVictims)$
        $uncapValue \leftarrow CappedVictims[victim.Host] * 2$
        $Uncap(victim, uncapValue)$
        $CappedVictims[victim.Host] \leftarrow uncapValue$
        if victim.curCap $== 100$ then
            $delete(CappedVictims, victim.Host)$
        end if
    end if
end procedure
```

Each node runs a 64-bit Linux 4.4.0-64 kernel and shares an NFS server. Apache Mesos 1.1.0 is deployed as the cluster manager. The Electron framework is used as the sole Mesos framework to schedule workloads. Docker 1.13.1 is used as the container technology. Benchmarks are run inside Docker containers to ensure environment consistency across worker nodes. Performance Co-Pilot [55] is deployed across the cluster to collect metrics from all worker nodes. Metrics collected from worker nodes include energy measurements from RAPL[1] counters and various statistics about CPU and memory usage from each worker node's Linux kernel. No metrics are collected from our Master nodes as they do not run any workload and thus have limited impact on variable power and energy consumption.

5.3.2 Workloads

The benchmarks with which we created our Light, Moderate, and Heavy Power Consuming Workloads were derived from the DaCapo Benchmark suite [6], Phoronix Benchmark suite [57], MiniFE from Mantevo [49], and Stream and Dgemm from NERSC [53]. Benchmarks like HiBench[26] were not used as the current focus is only on benchmarks that are designed to run on a single node in the cluster.

5.3.3 Median of Medians Max Power Usage

There are many ways of calculating a suitable global value to be used as an estimation of the power consumption for each benchmark. For this set of experiments we opted to use the Median of Medians of the Max Power Usage (M^3PU) value for each benchmark as an approximation of the power consumption in our workloads (described in Algorithm 5).

Since our cluster is heterogeneous, the estimated values varied between machines belonging to the four different power classes described in Section 5.3.1. For each

[1]RAPL only supports monitoring CPU and DRAM. Thus, any references to power and energy should be understood to mean energy consumed by CPU and DRAM.

Test suites	**Description**	**Type**
Audio Encoding*	Runtime measurement to encode WAV file to different audio formats.	CPU
Video Encoding†	Video encoding tests, processor tests and system performance testing.	CPU
Cryptography†	Cryptography tests such as OpenSSL and GnuPG.	CPU
Network Loopback*	Computer's networking performance testing.	Network
Avrora*	Multithreaded AVR microcontrollers simulator.	CPU
Batik*	Produces Scalable Vector Graphics images.	Memory
Eclipse*	Non-GUI jdt performance tests for the Eclipse IDE.	CPU
Jython*	Interprets the pybench Python benchmark.	CPU
Pmd†	Multithreaded Java source code analysis.	CPU
Tradebeans*	Daytrader benchmark run on GERONIMO with an in-memory H2 DB.	Memory
H2*	Executes transactions against a model of a banking application.	Memory
Xalan†	Multithreaded XML to HTML converter.	Mixed
Sunflow†	Renders a set of images using ray tracing.	CPU
miniFE[49]*	Finite element generation, assembly and solution for an unstructured grid problem.	CPU
DGEMM[62]†	Multi-threaded, dense-matrix multiplication.	CPU
STREAM[31]*	Calculates sustainable memory bandwidth and the computation rate for simple vector kernels.	Memory

Table 5.1: Workload benchmarks
The $\dagger$ symbol indicates a High Power Consuming benchmark while the $\star$ symbol indicates a Low Power Consuming benchmark as determined through profiling and k-means clustering.

benchmark, ten profiling runs were recorded on four nodes, one for each class. The max peak was found for each of the ten runs. Each power class then had ten max peaks from which the median was calculated. From the median we subtracted the median static power for each power class, generating a Median Max Power Usage (MMPU) of the benchmark for each power class. We used these values as an approximation

Algorithm 5 Median Median Max Power Usage (M^{3}PU)

```
1:  $R \leftarrow$ Number of individual runs.
2:  $P \leftarrow$ Power.
3:  $Peaks \leftarrow$ Power peaks per run.
4:  $PC \leftarrow$ Power Classes.
5:  procedure M$^3$PU($Benchmarks$[...], $PC$[...])
6:      for bm in $Benchmarks$ do
7:          $MMPU \leftarrow List()$
8:          for pc in $PC$ do
9:              $peaks \leftarrow bm.getPeaks(pc)$
10:             $mmpuPc \leftarrow$ BENCHMARK_MMPU($peaks, pc$)
11:             $MMPU[...] \leftarrow mmpuPc$
12:         end for
13:         $M^3PU[bm] \leftarrow Median(MMPU[...])$
14:     end for
15: end procedure
16: procedure BENCHMARK_MMPU($Peaks[R][P]$, $PC$)
17:     $MaxPeaks \leftarrow List(R)$
18:     for i in 0 to R-1 do
19:         $MaxPeaks[i] \leftarrow$ MaxPeak(Peaks[i])
20:     end for
         return $Median(MaxPeaks)$ - $StaticPower_{PC}$
21: end procedure
```

of the worst case power consumption of the benchmark on any node in that power class. The four MMPU values were used as observations for our task classification described in Section 5.3.4.

In order to be able to build the data structures required for our scheduling policies we required a single value as a point of comparison for sorting. We opted to use the Median of the four MMPU values which represents a cluster-wide central tendency of power usage for each benchmark, resulting in a Median of Medians Max Power Usage (M^{3}PU) for each benchmark.

5.3.4 Task Classification

Using the well known *k-means* clustering algorithm with the four MMPU values of each benchmark as observations, we classified benchmarks into two categories: low power consuming and high power consuming. As a benchmark can be scheduled on any node in the cluster, the power consumption of the benchmark on different power classes needs to be considered. For this reason, the MMPU values for each power class were used as the observations instead of the global approximation M^3PU value. The classification of our benchmarks can be seen in Table 5.1. Using this classification we created three kinds of workloads: Light, Moderate, and Heavy. Each workload has a different ratio of low power consuming tasks to high power consuming tasks: Light (20:3), Moderate (10:6), and Heavy (5:12).

5.4 Performance Analysis

In this section, we analyze the performance of different strategies for consuming Mesos offers: *First-Fit (FF)*, *Bin-Packing (BP)*, *Max-Min (MM)*, and *Max-GreedyMins (MGM)*. We also study the effect of two power capping strategies, *Extrema* and *Progressive Extrema*, when used with each scheduling policy. To further discover strengths and weaknesses of a combination, we run three workloads: Light, Moderate, and Heavy. We quantify each combination of scheduling policy, power capping strategy, and class of workload based upon the following aspects: (1) ability to reduce peak power consumption, (2) ability to reduce energy consumption, and (3) impact on makespan.

5.4.1 Performance of Scheduling Policies

Tables 5.2, 5.3, and 5.4 compare the energy, makespan, and the 95th percentile in power consumption, for different scheduling policies for a Light, Moderate, and Heavy Power Consuming Workload respectively.

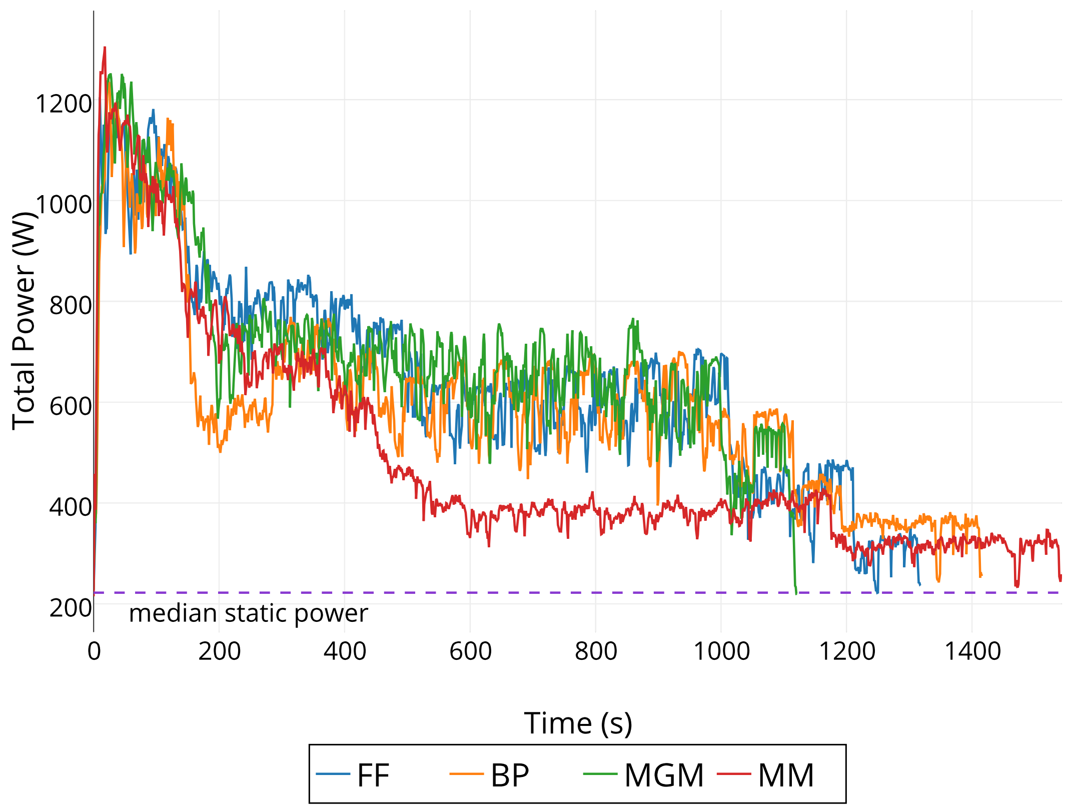

Figure 5.5: Power consumption of Light Power Consuming Workload when scheduled with different scheduling policies.

	FF	**BP**	**MM**	**MGM**
Power (W)	1072.75	1033.1	1039.2	1094.8
Makespan (s)	1319	1417	1544	1122
Energy (kJ)	858	844.6	761.9	793

Table 5.2: Comparison of the effects of different scheduling policies for a Light Power Consuming Workload.

Light Power Consuming Workload

Figure 5.4.1 shows the power profiles of the execution of the Light Power Consuming Workload (LPCW) when using the previously mentioned scheduling policies.

Power: Both BP and MM experience improvements in the 95th percentile of power consumption when compared to FF and MGM, where BP slightly improves over MM by 6.1 Watts. The low power envelope for MM can be attributed to the fact that a larger number of high power consuming tasks complete execution earlier, leaving behind only low power consuming tasks running on the cluster.

Makespan: BP and MM experience a significant increase in makespan when compared to FF. The increase in makespan for BP can be attributed to excessive co-location of high power consuming tasks at a later execution stage, leading to an increase in resource contention for some nodes. Although MM address one of BP's shortcomings by scheduling high power consuming tasks earlier in the scheduling process, it suffers an increase in makespan as a result of delaying of the execution of many of the low power consuming tasks, leading to decreased throughput. From Table 5.2, we can see that MGM experiences a marked improvement in makespan when compared to FF, BP, and MM. MGM achieves this improvement by reducing the co-location of high power consuming tasks while concurrently consuming an increased amount of low power consuming tasks, thus being particularly beneficial for the ratio of low power consuming tasks to high power consuming tasks in the LPCW.

Energy: Although BP shows a slight improvement in energy consumption when compared to FF, the impact on makespan incurs a severe static power penalty. On the other hand, MM and MGM show a significant reduction in energy consumption when compared to BP and FF. Furthermore, MM experiences a 31.1 kJ reduction in energy consumption when compared to MGM. Although MM incurs a makespan penalty, it achieves this low energy consumption by maintaining a low power envelope. In contrast, MGM does not have a huge impact on the power peaks, but the significant reduction in makespan leads to an improvement in energy consumption as it avoids static power penalties.

Moderate Power Consuming Workload

	FF	**BP**	**MM**	**MGM**
Power (W)	1032.9	957	980	1049.4
Makespan (s)	1521	1602	1575	1450
Energy (kJ)	1183.8	1031	918.3	891.7

Table 5.3: Comparison of the effects of different scheduling policies for a Moderate Power Consuming Workload.

Figure 5.6 shows the power profiles of the execution of the Moderate Power Consuming Workload (MPCW) using the previously mentioned scheduling policies.

Power: Table 5.3 shows an improvement in the 95th percentile of power consumption for BP when compared to FF, MM, and MGM. However, MGM experiences a substantial reduction in the 90th percentile (not shown in the table) of power consumption, improving over FF, BP, and MM by 304 Watts, 88.8 Watts, and 39.8 Watts

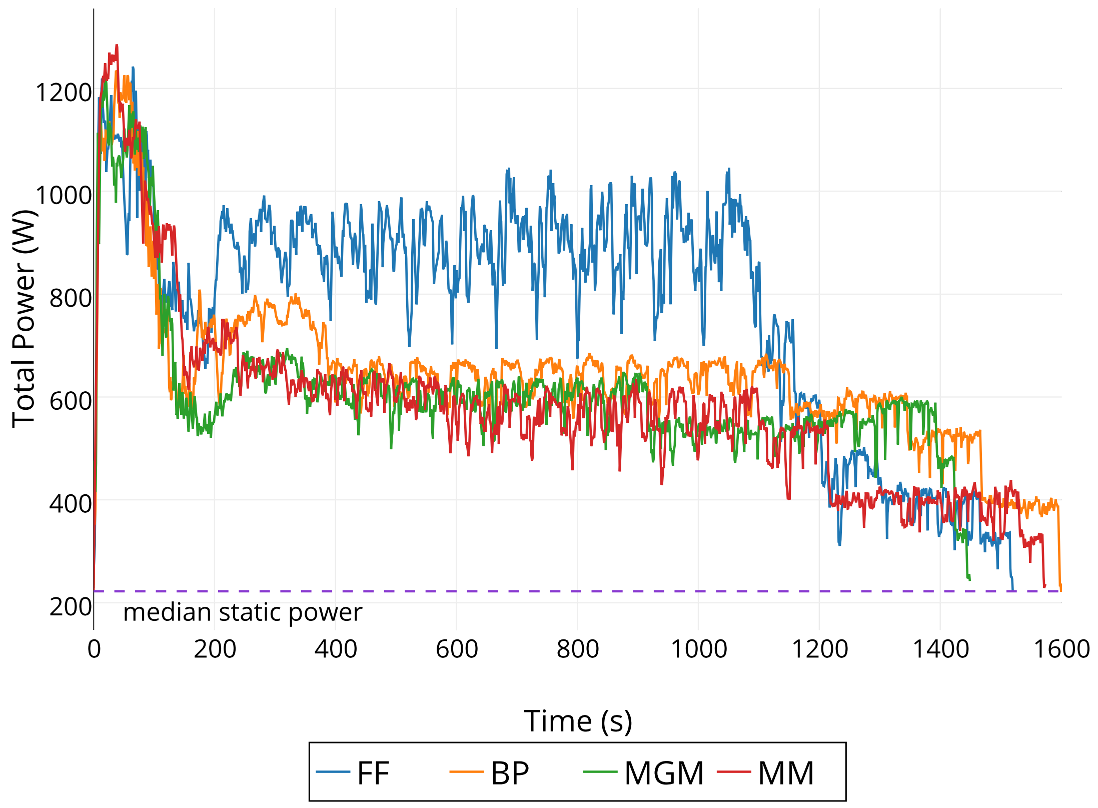

Figure 5.6: Power consumption of Moderate Power Consuming Workload when scheduled with different scheduling policies.

respectively. This discrepancy between 90th and 95th percentile in power consumption for MGM can be attributed to early execution of high power consuming tasks, leading to a high initial spike of power consumption. Throughout the rest of execution, MGM maintains a lower power profile in comparison to the other scheduling policies.

Makespan: BP suffers an increase in makespan when compared to FF, MM, and MGM, which can be attributed to BP co-locating several high power consuming tasks late in the task allocation process. This excessive co-location leads to an increase in contention for resources, thus increasing the completion times for these high power consuming tasks. MGM's reduction in makespan can be attributed to better distribution of high power consuming tasks across the worker nodes.

Energy: Although BP reduces the power envelope, the increase in makespan reduces the impact it has on the energy consumption due to the static power penalty. However, BP still consumes 152.8 kJ less than FF. On the other hand, MM and MGM are able to achieve a more heterogeneous mix of low power consuming and high power consuming tasks, thereby reducing energy consumption. As MGM results in a further increase in the distribution of high power consuming tasks across the cluster, it experiences a 26.6 kJ reduction in energy consumption when compared to MM.

	FF	**BP**	**MM**	**MGM**
Power (W)	1098.2	1006.5	1110.7	1028.2
Makespan (s)	1630	1683	1626	1697
Energy (kJ)	1546.4	1380.1	1259.1	1226.9

Table 5.4: Comparison of the effects of different scheduling policies for a Heavy Power Consuming Workload.

Heavy Power Consuming Workload

Figure 5.7 shows the power profiles of execution of the Heavy Power Consuming Workload (HPCW), using various scheduling policies.

Power: As the HPCW contains an increased number of high power consuming tasks, we can see a clear increase of power envelopes for all the policies. Table 5.4 shows that BP experiences a substantial reduction in the 95th percentile of power consumption when compared to FF. Furthermore, BP is better than MM by 104.2 W in the 95th percentile of power consumption. Although MM improves the distribution of tasks

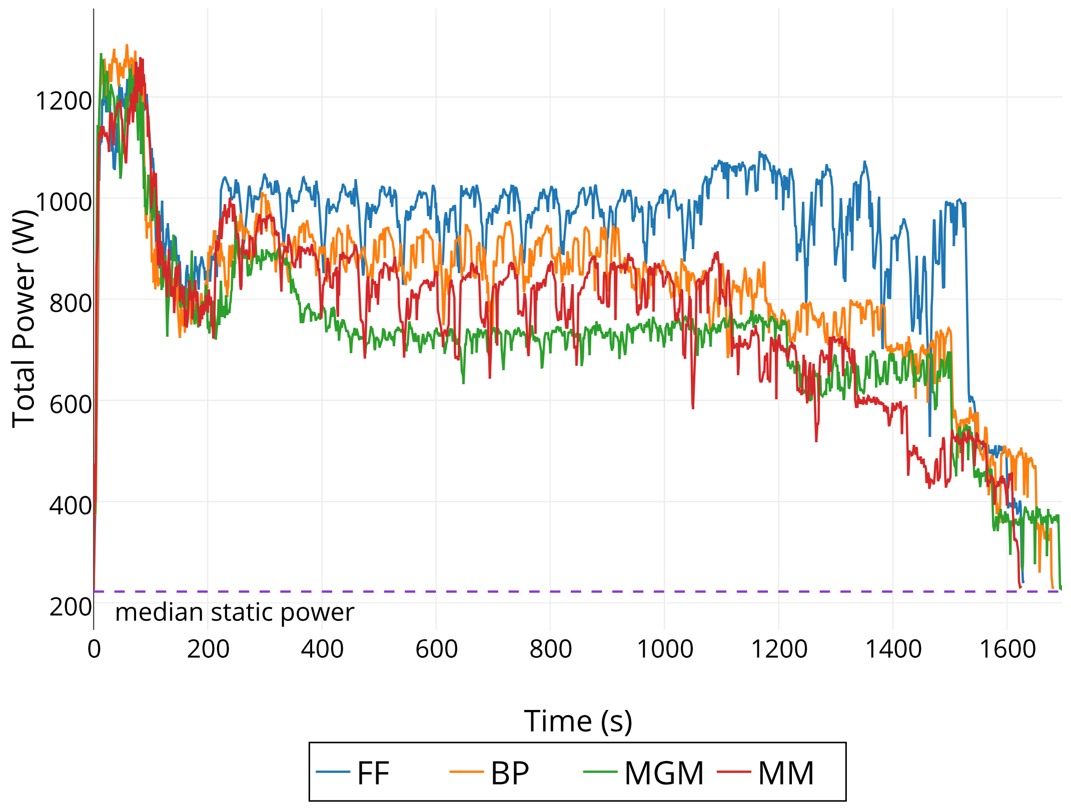

Figure 5.7: Power consumption of Heavy Power Consuming Workload when scheduled with different scheduling policies.

across the cluster, it does not have a substantial impact in reducing the excessive co-location of high power consuming tasks for this power intensive workload. MGM, however, shows an improvement in the 95th percentile when compared to FF, BP, and MM, and this reduction can be attributed to a decrease in the co-location of high power consuming tasks, leading to a reduction in coincident peaks.

Makespan: Although MGM shows an improvement in power consumption, it delays the start time of execution of the high power consuming tasks. This increase in latency for power intensive tasks leads to an increase in makespan, as seen in the data shown in Table 5.4. In addition, as the workload gets more power intensive, MGM's detrimental impact on makespan might become more prominent. MGM shows a similar makespan to BP, posting just a 14 second difference, but the increase in makespan for BP can be attributed to resource contention of excessively co-located high power consuming tasks.

Energy: BP experiences a significant reduction in energy consumption when compared to FF, as seen in Table 5.4. Furthermore, MM and MGM experience a decrease in energy consumption when compared to BP. Although MM experiences a slight increase in energy consumption of around 32 kJ when compared to MGM, MM would be a more appropriate choice when scheduling a higher ratio of high power consuming tasks as MGM is more likely to have a negative impact on makespan as the ratio of high power consuming benchmarks to low power consuming benchmarks increases. Eventually, the increase in makespan would lead to the static power penalty nullifying energy decrease from maintaining a lower power envelope.

5.4.2 Power Capping Impact On Scheduling

In this section we quantify the impact of our set of Power Capping Strategies {Extrema, Progressive Extrema} across our set of different scheduling policies, {FF, BP, MM, MGM}.

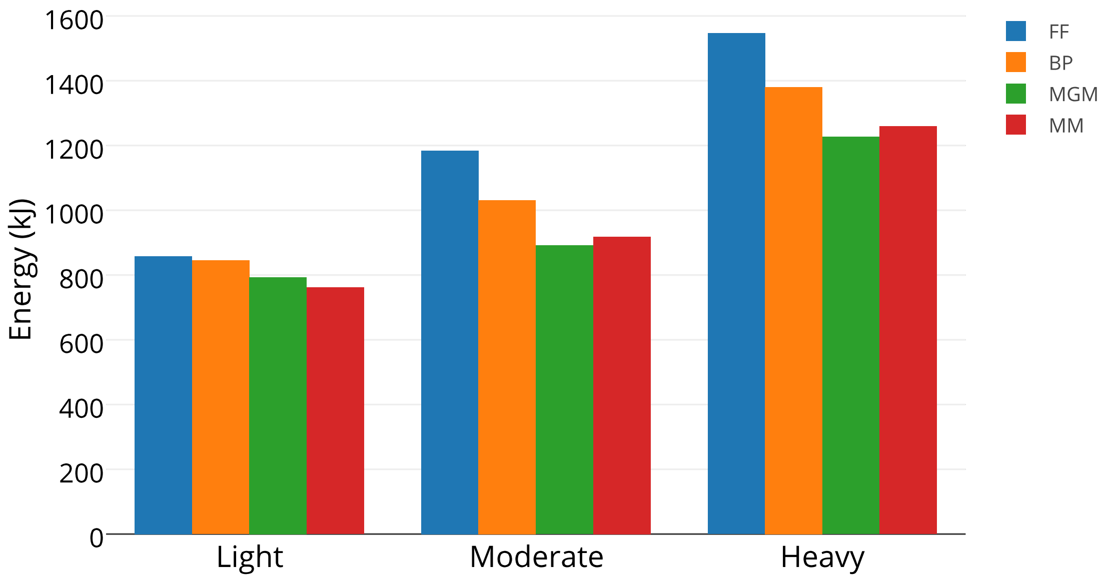

Figure 5.8: Uncapped workloads. Comparison of Cluster-wide Energy Consumption of different scheduling policies when used to schedule different classes of workloads (Light, Moderate and Heavy).

Extrema

Figures 5.11 to 5.13 show the effect of using the Extrema power capping strategy when used alongside different scheduling policies for the Light, Moderate, and Heavy Power Consuming Workloads (LPCW, MPCW, and HPCW respectively).

Power: Figure 5.11 shows the power profiles when Extrema is run alongside the different scheduling policies for the LPCW. Compared to their uncapped runs, FF, BP, MM, and MGM experience a reduction in the 95th percentile of power consumption

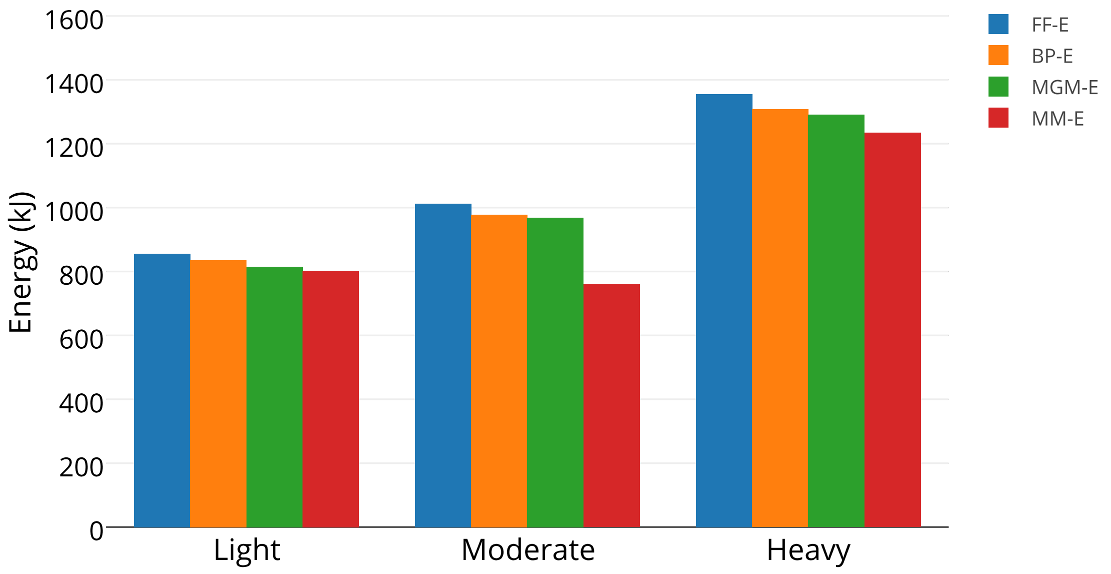

Figure 5.9: Extrema capped workloads. Comparison of Cluster-wide Energy Consumption for different scheduling policies and classes of workloads in combination with the Extrema dynamic power capping strategy.

of 231 Watts, 190 Watts, 193 Watts, and 251 Watts, respectively, when running under the Extrema capping policy. For the MPCW, shown in figure 5.12, FF, BP,and MGM experience an improvement of 220 Watts, 74 Watts, and 7 Watts, respectively, in the 90th percentile of power consumption compared to the uncapped runs. On the other hand, MM with Extrema experiences an increase of 23 Watts in the 90th percentile of power consumption for the MPCW compared to its uncapped run. The effect of Extrema on MM is likely due to the size of tasks in the *deque* and the time Extrema needs to place and adjust power caps on each node. When all higher energy

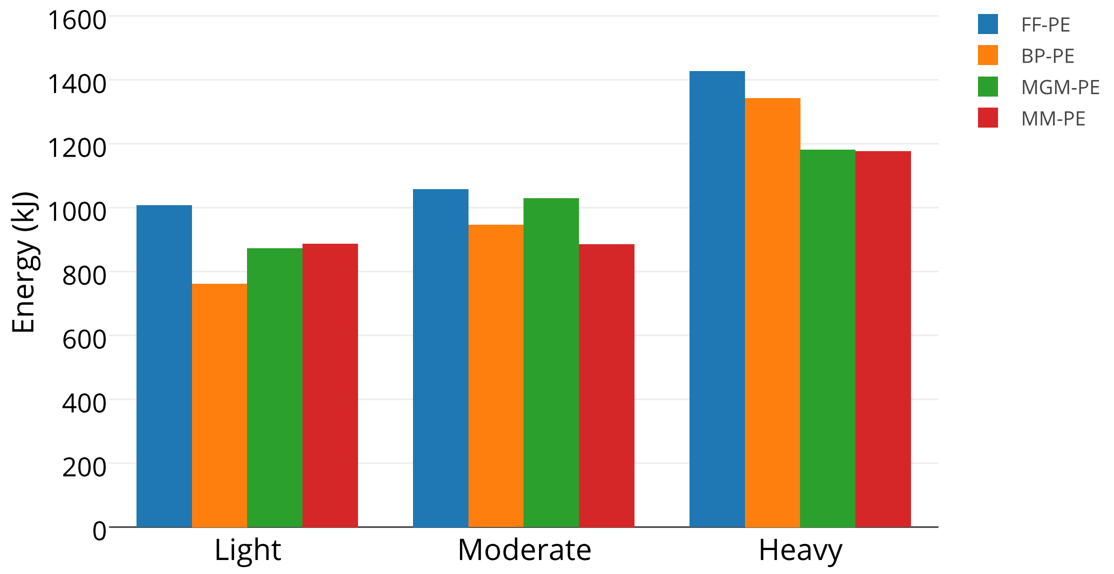

Figure 5.10: Progressive Extrema capped workloads. Comparison of Cluster-wide Energy Consumption for different scheduling policies and classes of workloads in combination with the Progressive Extrema dynamic power capping strategy.

jobs remain in the *deque* after the lower energy jobs have been exhausted, the power capping gets more aggressive and the cluster remains capped for a longer period of time. More investigation into the exact circumstances that lead to a power increase in this instance is the subject of future work. When scheduling the HPCW, the scheduling policies FF, BP, MM, and MGM show an improvement in the 90th percentile of power consumption (relative to their uncapped runs) of 118.5 Watts, 94.2 Watts, 116 Watts, and 35 Watts respectively.

Makespan: Extrema does not impact the makespan of FF as the workload is well

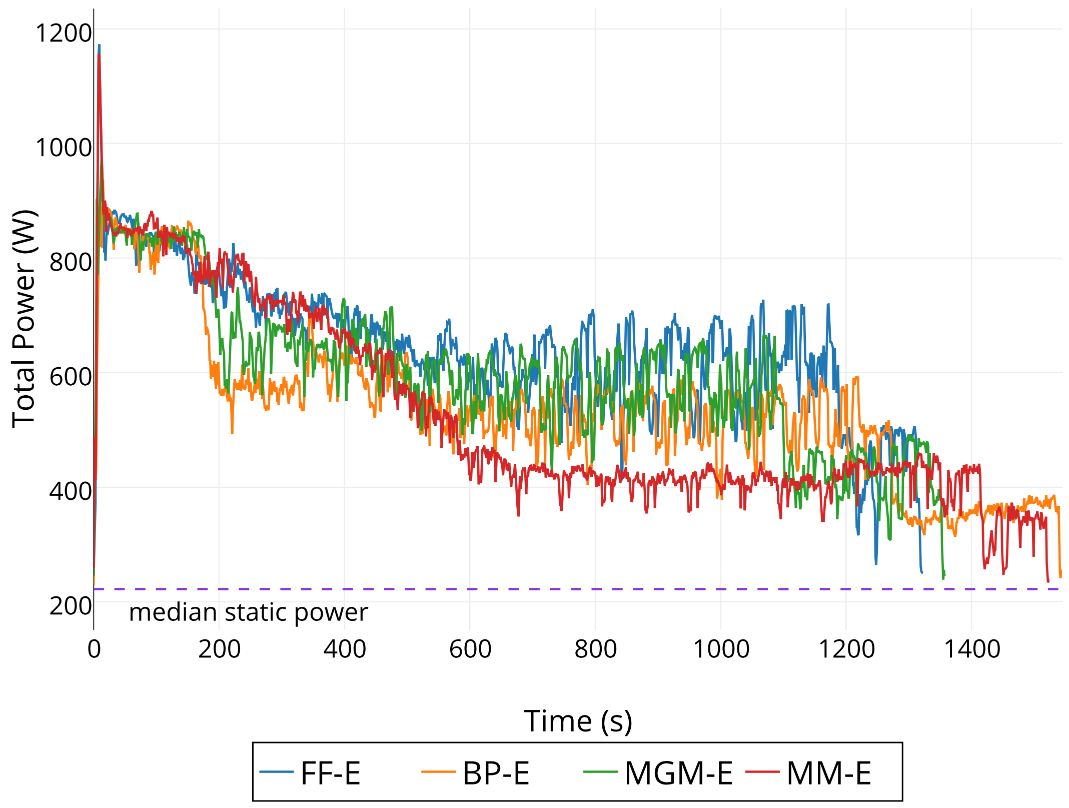

Figure 5.11: Power profile of Light Workload run under Extrema capping.

distributed across the cluster. However, when Extrema is used alongside BP, the makespan is increased by 128 seconds, 67 seconds, and 76 seconds for LPCW, MPCW, and HPCW respectively. When Extrema is used alongside MM for the LPCW and MPCW, the makespan is not affected. On the other hand, when Extrema is used alongside MM for the HPCW, it experiences an increase in makespan of 94 seconds. This increase is due to RAPL lowering CPU clock speeds to stay within a power budget which has an adverse effect on the larger number of high power consuming tasks in contention for system resources. When Extrema is used alongside MGM,

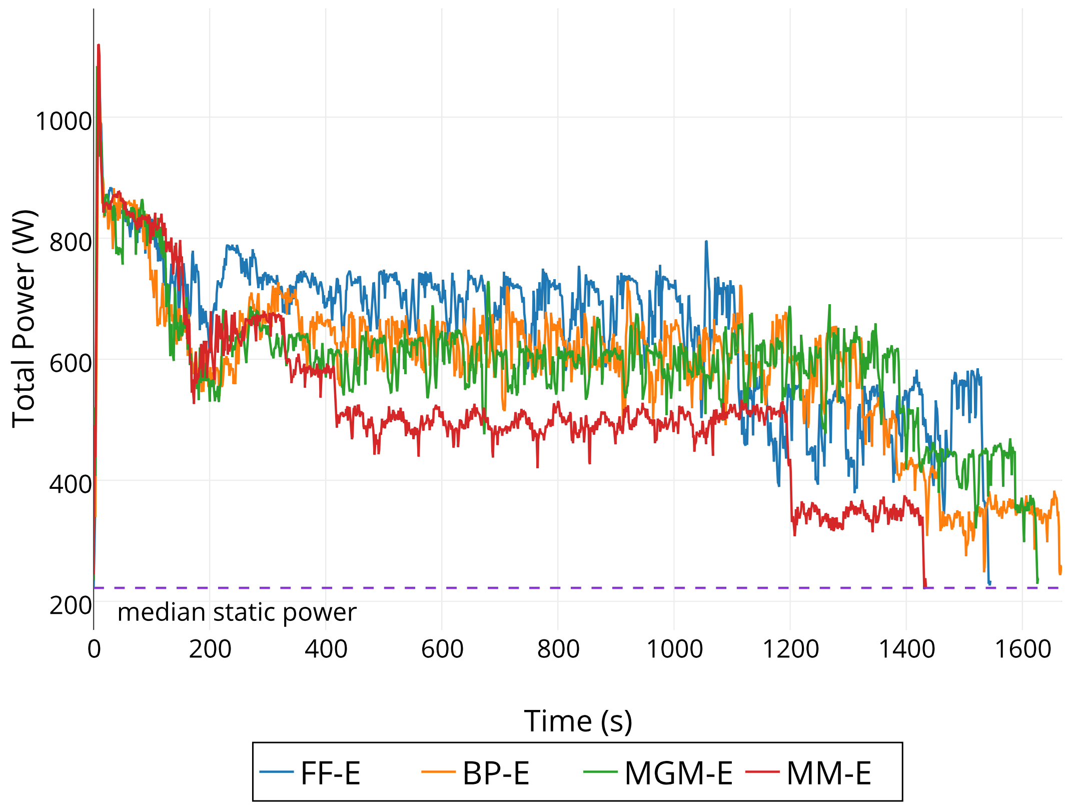

Figure 5.12: Power profile of Moderate Workload run under Extrema capping.

there is an increase in makespan of 236 seconds for the LPCW, and 179 seconds for the MPCW. This indicates that Extrema combined with MGM is not a good fit for systems that want to maintain a high Service Level Agreement (SLA) for processing LPCW and MPCW. There is no impact on the makespan when Extrema is used with MGM for the HPCW, shown in figure 5.13, as MGM is better at distributing the high power consuming tasks across the cluster as compared to BP and MM.

Energy: In general, Extrema's reduction in peak power consumption is much more prominent than Extrema's increase in makespan. This leads to Extrema lowering the

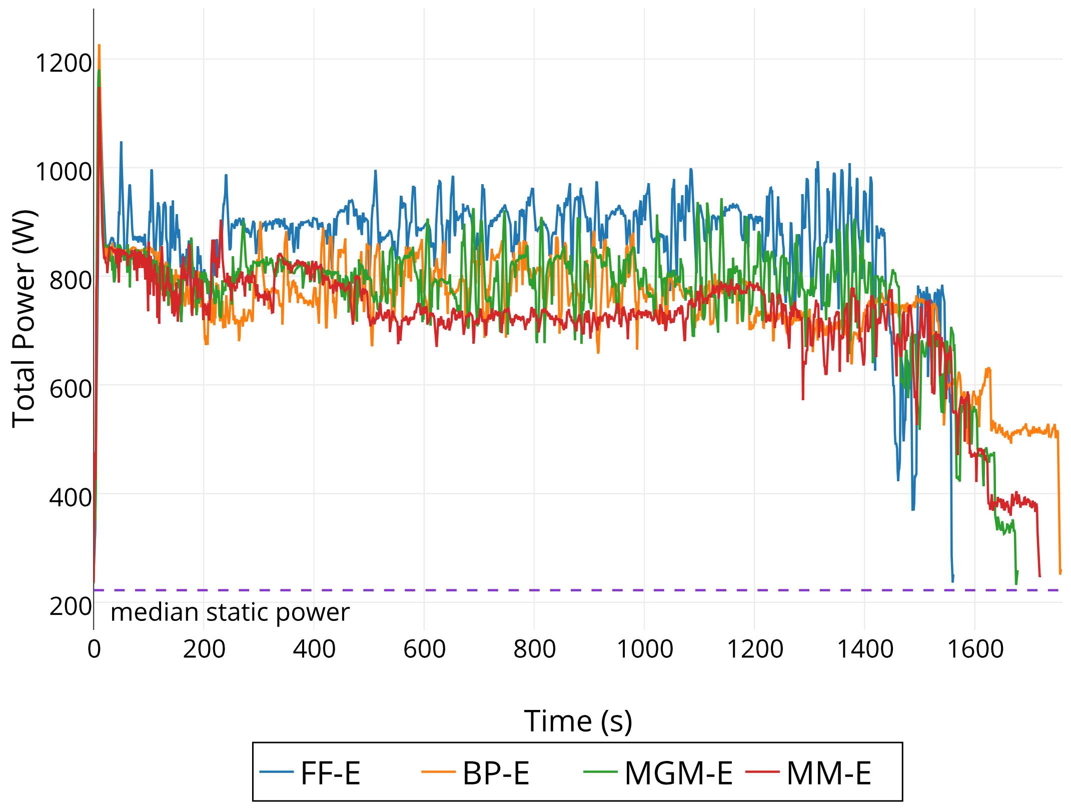

Figure 5.13: Power profile of Heavy Workload run under Extrema capping.

energy consumption for FF, BP, MM, and MGM for MPCW and HPCW. Figure 5.9 shows that when Extrema is used for the MPCW, the scheduling policies FF, BP, and MM experience a reduction in energy consumption of 171kJ, 53kJ, and 159kJ respectively when compared to their uncapped runs, while MGM, experiences an increase of 77 kJ in energy consumption compared to its uncapped run. When Extrema is used for the HPCW, our results show that compared to their uncapped runs, FF, BP, and MM experience a reduction of 191kJ, 73 kJ, and 25 kJ respectively, while MGM experiences an increase of 24kJ. The increase in energy consumption for MGM

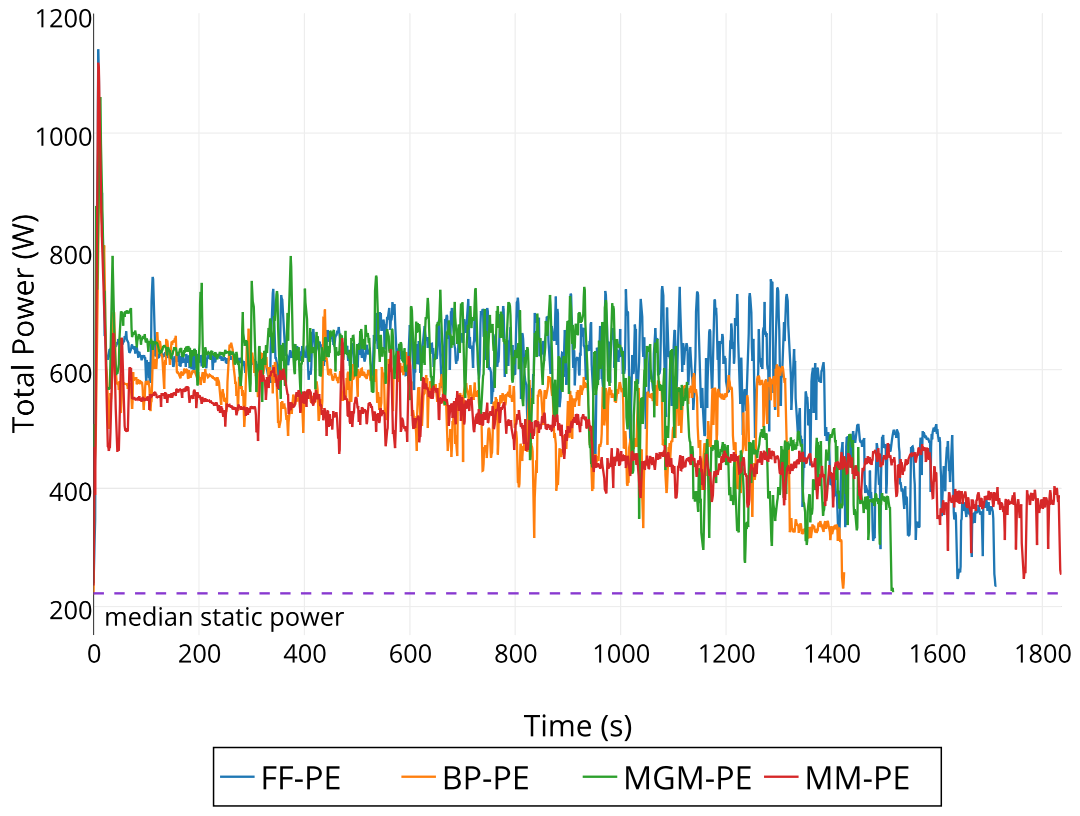

Figure 5.14: Power profile of Light Workload run under Progressive Extrema capping.

for MPCW as well as HPCW can be attributed to the delayed start time for the high power consuming tasks, thus incurring a heavier static power penalty.

Progressive Extrema

The graphs in Figures 5.14 to 5.16 show the effect of using the Progressive Extrema power capping strategy alongside the scheduling policies described above for the Light (LPCW), Moderate (MCPW), and Heavy (HPCW) Power Consuming Workloads.

Power: Compared with the uncapped runs and against the Extrema capped runs,

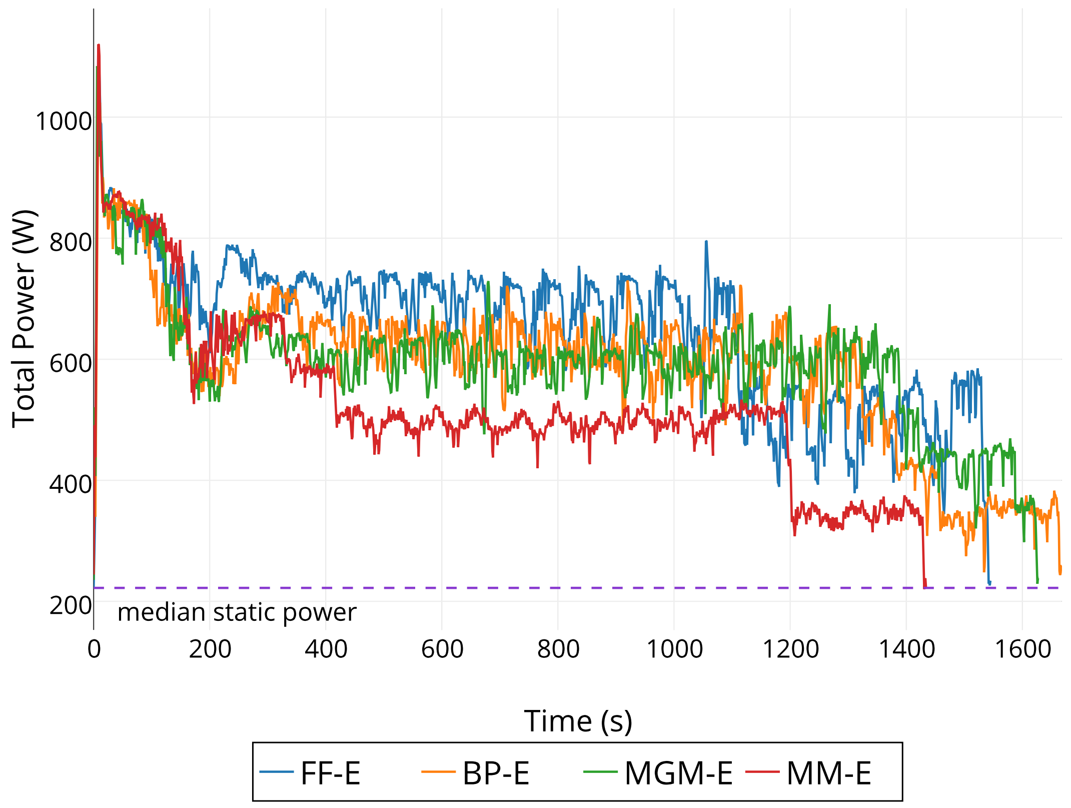

Figure 5.15: Power profile of Moderate Workload run under Progressive Extrema capping.

Progressive Extrema reduces the initial power draw on a cluster. The initial power peaks are around 1200 Watts for the uncapped runs, around 830 Watts for the Extrema power capped runs, and have been reduced to around 600 Watts for the LPCW and MPCW, shown in Figures 5.14 and 5.15. This reduction can be attributed to Progressive Extrema being able to more aggressively cap the already capped nodes, thereby quickly bringing down the power envelope closer to the predefined high and low thresholds. Analyzing the results shown in Figures 5.14 to 5.16, we observe

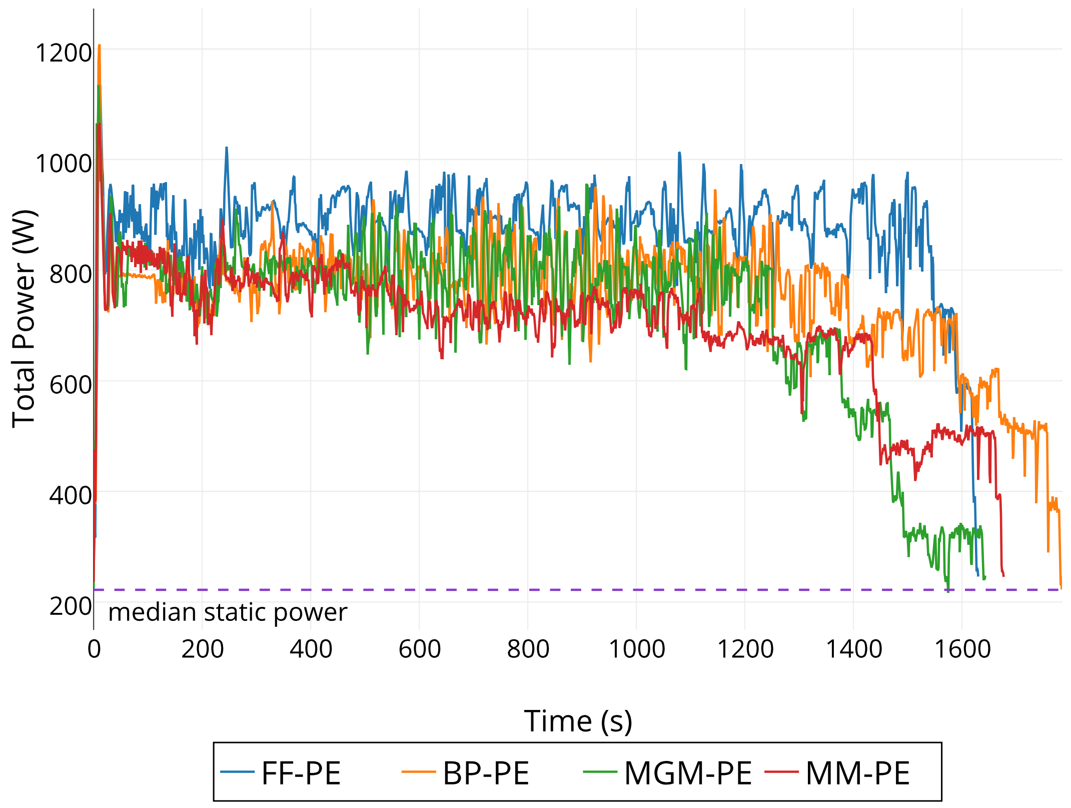

Figure 5.16: Power profile of Heavy Workload run under Progressive Extrema capping.

that when compared to their corresponding uncapped runs, shown in Figures 5.4.1, 5.6, and 5.7; FF, BP, MM, and MGM experience a substantial reduction in the 95th percentile (p95) of power consumption. FF experiences a p95 reduction in power consumption of 368, 374, and 141 Watts for LPCW, MPCW, and HPCW respectively. BP experiences a p95 reduction in power consumption of 405, 297, and 133 Watts for the LPCW, MPCW, and HPCW respectively. MM experiences a p95 reduction in power consumption of 457, 260, and 251 Watts for the LPCW, MPCW, and HPCW respectively. Finally, MGM experiences a p95 reduction in power consumption of 387, 401, and 155 Watts for the LPCW, MPCW, and HPCW respectively.

Makespan: Progressive Extrema has a negative impact on makespan as it is aggressive in power capping the nodes. For the LPCW, FF, MM, and MGM experience an increase in makespan of 394, 293, and 396 seconds, respectively, when compared to their corresponding energy consumptions shown in Figure 5.8. However, BP does not experience a significant impact on makespan for the LPCW when compared to its uncapped run. FF, BP, MM, and MGM experience an increase in makespan of 336, 53, 189, and 341 seconds, respectively, for the MPCW, when compared to their corresponding uncapped runs. BP and MM experience an increase in makespan of 102 and 52 seconds, respectively, for the HPCW. FF and MGM, however, due to a more even distribution of high power consuming tasks, do not experience an impact in makespan for the HPCW.

Energy: Although Progressive Extrema proves to be beneficial in significantly reducing the power envelopes and reducing the power fluctuations, the significant impact on makespan leads to Progressive Extrema not having a substantial improvement in energy consumption when compared to uncapped runs.

5.5 Conclusion

From our results we conclude that there are trade-offs that must be made in order to operate a cluster under a specific set of constraints. Some policies favor lowering the height of power peaks in the cluster, while others favor a reduced makespan, still others favor lower energy consumption, and any combination in between is also permissible. Although Max-Min and Max-GreedyMins often outperform First-Fit and Bin-Packing, we acknowledge these scheduling policies may cause starvation for tasks in workloads that are somewhere between Light and Heavy Power Consuming. Guarding against task starvation is not addressed in this paper and will be the subject of future work. Based upon our findings we have developed a few heuristics to help owners of Mesos-powered Data Centers schedule toward some of the goals presented above:

- Using an incorrect combination of scheduling policy and power capping strategy can lead to undesired outcomes such as increased energy consumption and larger makespans.
- Max-GreedyMins should be used when a workload requires high throughput and the scheduling time does not fall within Coincidence Peak, regardless of the power intensity of the workload.
- When the workload consists of a higher proportion of high power consuming tasks, scheduling policies, such as Max-Min, would be the more appropriate choice. On the other hand, if the workload consists of a higher proportion of low power consuming tasks, then scheduling policies, such as Max-GreedyMins, would be the more appropriate choice.
- To decrease power peaks and energy consumption, Extrema is best deployed as the power capping strategy to schedule Light and Moderate Power Consuming workloads while Progressive Extrema is best suited for Heavy Power Consuming workloads.

In our future work, we plan to develop a policy switcher that can switch between different scheduling policies and power capping strategies, so as to be able to handle a continuously changing workload.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Compared to First Fit scheduling strategies, Bin Packing variants provide various levels of reduced energy and power consumption. Energy consumption improvements are due longer makespan in cases where resource fragmentation prevents tasks with high resource requirements from starting their execution. This leads to an increase in static power consumption, a penalty incurred for simply maintaining the cluster online, to have an adverse effect on the overall Watts necessary to finish a workload.

Of the Bin Packing variants, each has strengths and weaknesses that make each better suitable for different workloads. For heavy power consuming workloads, a policy of Max Min is best suited to reduce energy and power consumption while maintaining an acceptable makespan. A workload with a higher quantity of low power consuming tasks will see benefit in the same metrics from using a Max Greedy Mins based scheduling algorithm.

Finally, Power capping a cluster without impacting workload makespan is difficult. When a node is capped, it is limited in how much work it is able to carry out due to RAPL forcing processors to maintain a higher P-state and thus a lower clock frequency. The three strategies presented in this thesis all have strengths and weaknesses. Static capping provides a safe guard against coincidence peak power, however, its impact on makespan has a detrimental effect on energy consumption due to static power consumption. Extrema presents a more fine tuned approach, making decisions to cap or uncap nodes based upon the recent power trends in the cluster. This leads to reduced power and energy consumption in most cases while having minimal impact on the time it takes to complete a batch of tasks. Progressive Extrema expands upon Extrema by providing several rounds of power capping, each more aggressive than the previous. This results in a lower peak power draw, sometimes at the cost of a longer makespan. However, even with a longer makespan, several workloads experienced a large enough of a decrease in power consumption such that the static power consumption penalty was overcome.

6.2 Future Work

6.2.1 Watts As A Resource (WaaR)

For every hardware component present in a node that provides resources traditionally accounted for, such as CPU, Memory, and Disk, there is a power consumption associated with its use. Power usage itself, however, is not usually among the resources accounted for that are necessary to perform a computation. Yet it is one of the toughest limiting factors to overcome in order for modern processors to improve

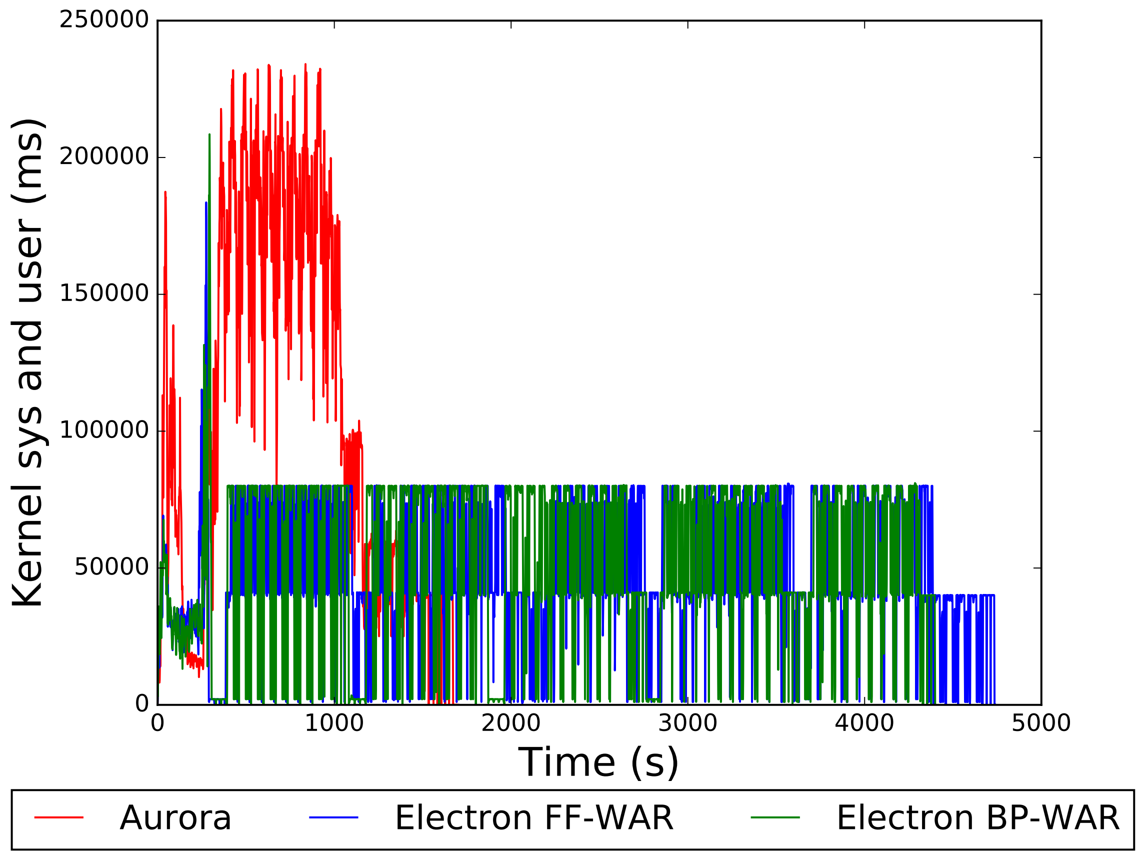

Figure 6.1: Cluster-wide CPU Utilization when WaaR is deployed and the cryptography benchmark is included. Electron FF-WaaR and Electron BP-WaaR use far less power and CPU time, but suffer a larger runtime.

[68]. In this thesis, we have looked at addressing the issue of co-incident peak power consumption and energy consumption through the use of static and dynamic power capping but we believe the most efficient way to address this problem is by promoting watts as a first class scheduling parameter. By introducing Watts as a Resource (WaaR) we are able to account for power in the same way other resources, such as CPU, Memory and Disk, are accounted for. As with CPU, Memory, and Disk, the closer our we are able to bring our theoretical accounting of Power consumption to

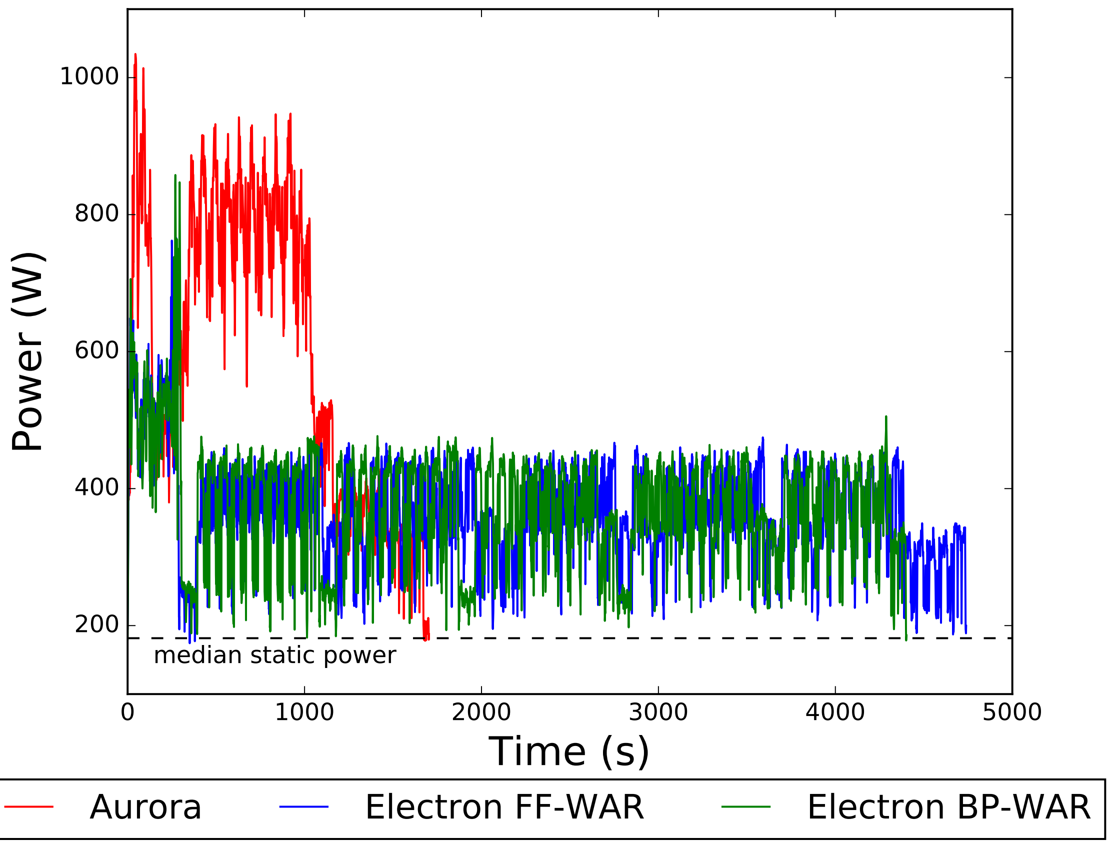

Figure 6.2: Cluster-wide Power Utilization when WaaR is deployed and the cryptography benchmark is included. The Power ceiling is lowered by the deployment of Watts as a Resource. Cryptography is the highest Power consuming benchmark in our set, causing it to starve when we decrease the number of Watts as a resource available in the cluster.

match that of the real world consumption, the more efficiently we are able to manage cluster wide power-consumption. Our initial experiments and results shown in Figures 6.1 to 6.3 show there may be some promise in developing this technique. In our experiments, we observed a big trade-off between the power ceilings and runtime. With WaaR enabled, Electron's Bin-Packed runtime is around 2.58 times as long as Aurora's runtime, while Electron First-Fit fared a bit worse at 2.78 times as long as Aurora's runtime. However, it is encouraging that the median power consumption

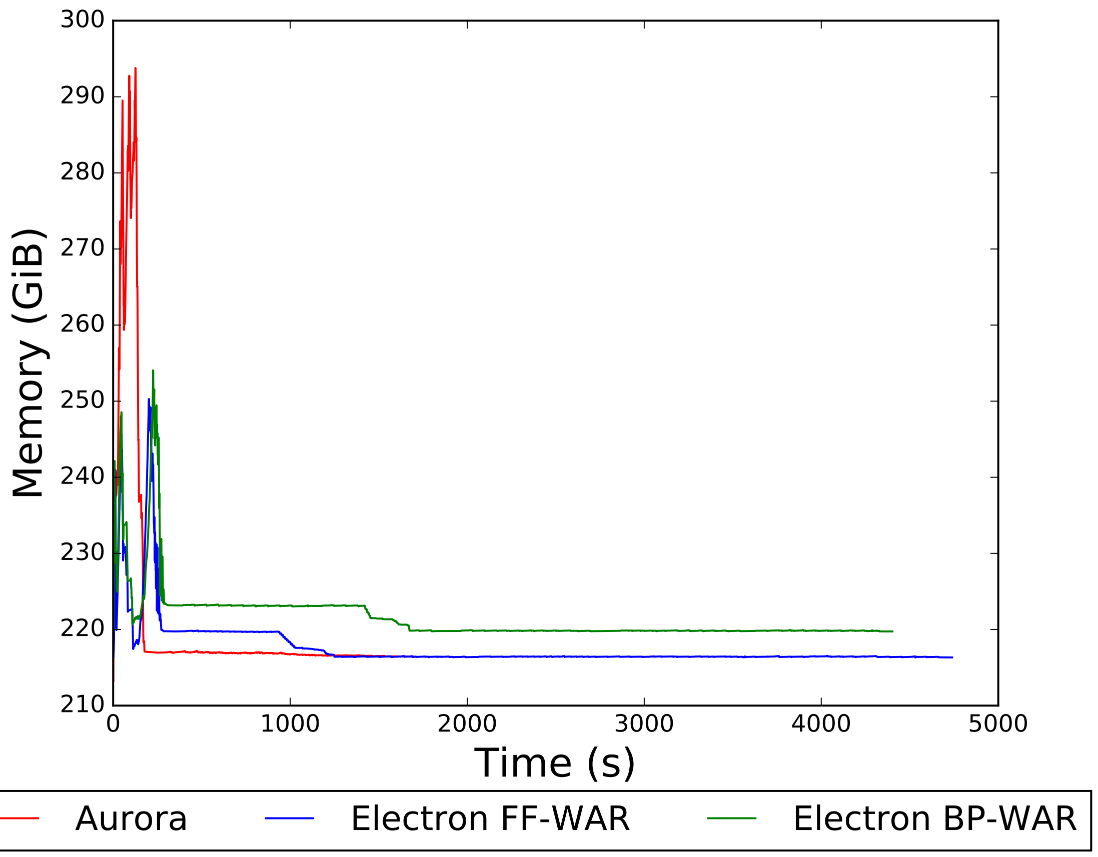

Figure 6.3: Cluster-wide Memory Utilization when WaaR is deployed and the cryptography benchmark is included.

for Electron First-Fit was around 43% and Electron Bin-Packing was around 51% of Aurora's median power consumption, while the peak power reduced by 30% for Electron First-Fit WaaR and 19% for Electron Bin-Packed WaaR.

An obvious downside of the WaaR approach is that workloads that are CPU intensive may starve as their power needs may only be met by a small subset of the Mesos offers. An example of such a case is the Cryptography benchmark, which is CPU intensive. The difference can be observed in Figure 6.5 where the same experiment seen in Figure 6.2 was repeated *without* the inclusion of Cryptography.

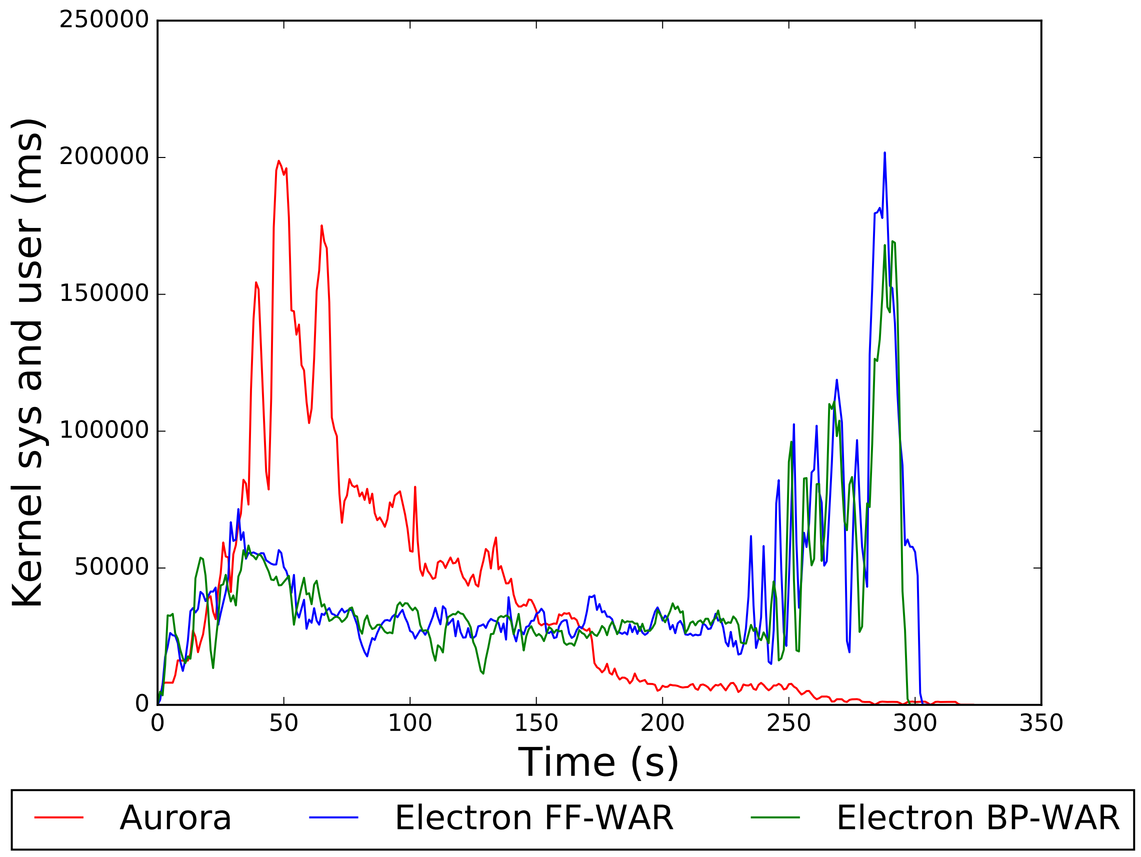

Figure 6.4: Cluster-wide CPU Utilization when WaaR is deployed but the cryptography benchmark is not included.

Watts as a Resource does better in this case by improving over Aurora in runtime and power ceiling. It is possible to identify stragglers in advance, so that they can be tagged and the watts limits can be relaxed, for exactly the windows at which such workloads have to be executed. Additionally, it is possible to use our experimental setup to determine the best co-runners with such workloads. While these numbers reflect a poor performance in terms of makespan, this is due to our naive way of accounting for Power utilization. In this case, we used the (M^3PU) of each benchmark to symbolize how much power it would be consuming once scheduled. As we have

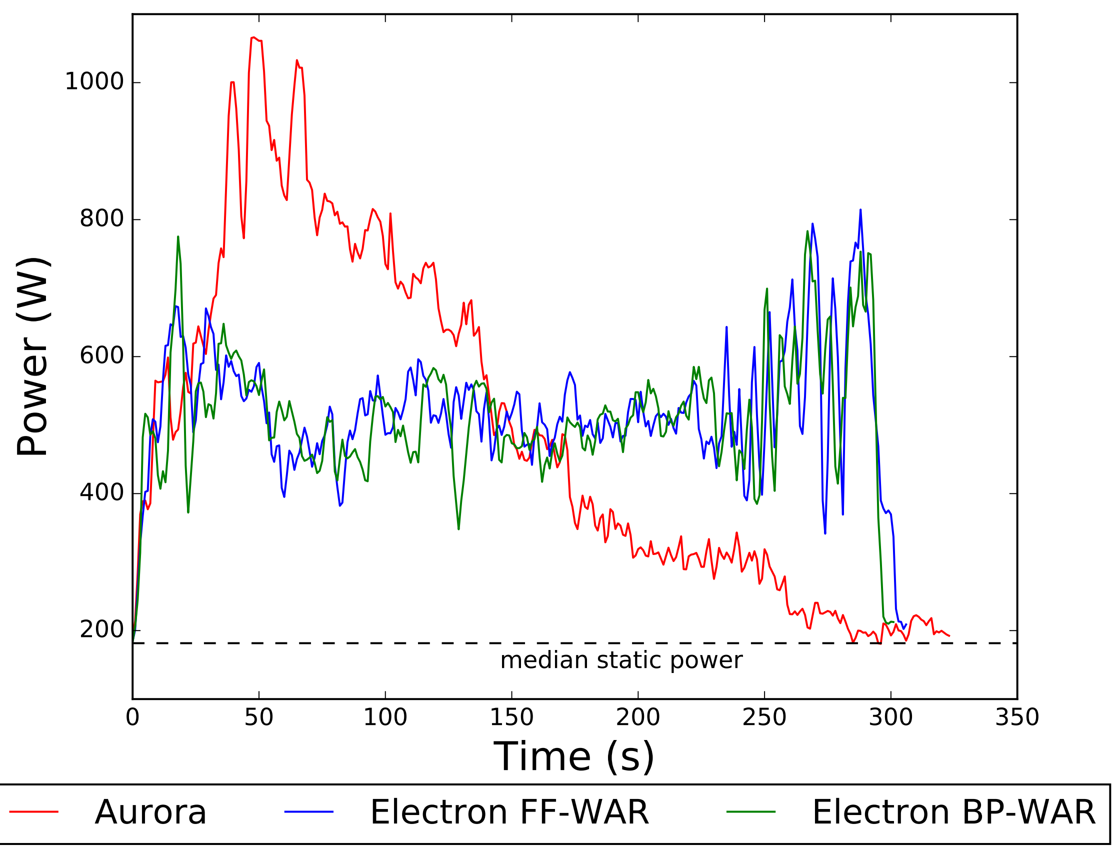

Figure 6.5: Cluster-wide Power Utilization when WaaR is deployed but the cryptography benchmark is not included.

demonstrated in this thesis,power consumption cannot be reduced to a sum of the individual benchmarks running on a node. Therefore, a more appropriate approach is a to approximate power consumption on a node given a set of workloads to be run on the node.

6.2.2 Adaptive Power Aware Scheduler

With our Bin Packing, Min Max, and Max-GreedyMins policies, there is a risk of starvation. For example, if we consider using Min Max and Max Greedy Mins as our

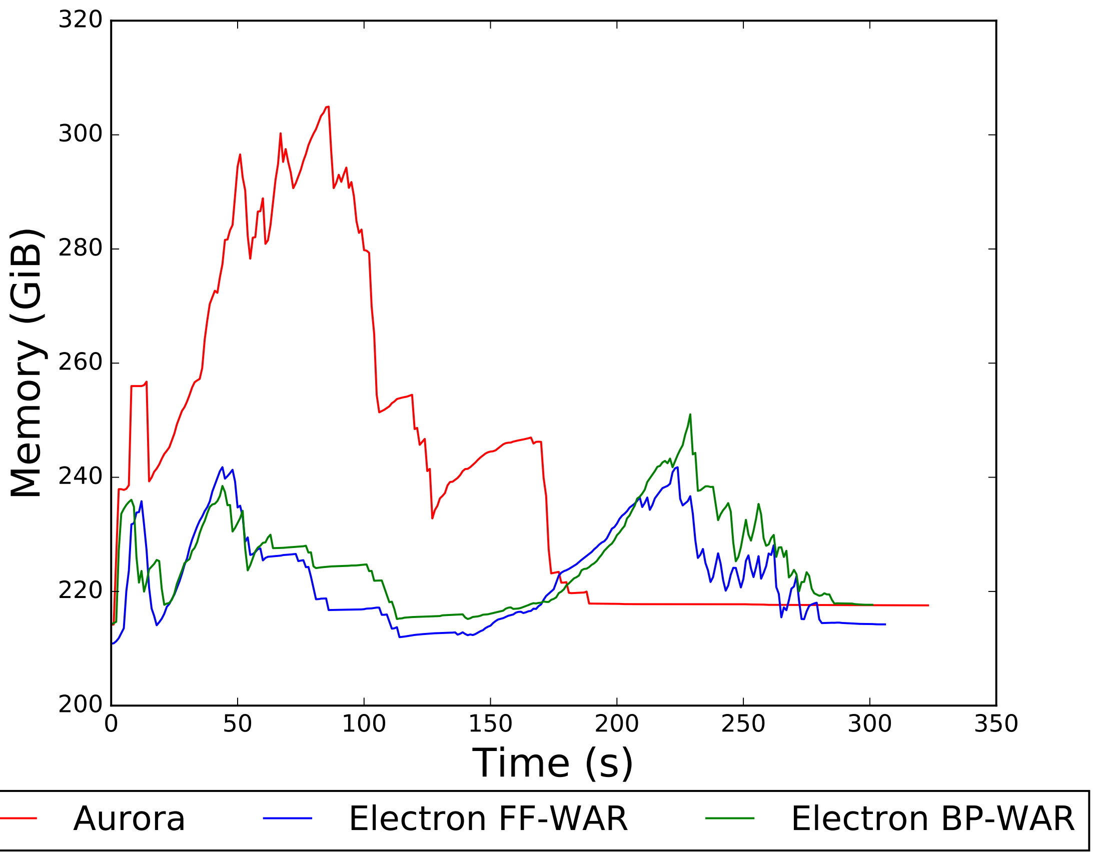

Figure 6.6: Cluster-wide Memory Utilization when WaaR is deployed but the cryptography benchmark is not included.

scheduling algorithms and assume a workload contains three classification of tasks A, B, and C. Class A tasks are high power consuming task, class B tasks are low power consuming task, and class C tasks consume less power than class A tasks but more than class B tasks. If there are enough class A and B tasks, class C tasks will not be scheduled until either all class A tasks are all scheduled or all class B tasks are consumed. Solutions to this drawback will have an impact on power and energy consumption, and makespan. Different techniques to mitigate starvation may be more beneficial to certain kinds of workloads.

Using the results from the experiments in this dissertation, it may be possible to create a scheduling policy that adapts to the power and energy consumption requirements for operating a cluster or the current demand from the workloads queued to run on the cluster.

6.2.3 CPI based Power Aware Scheduler

Processors are complex pieces of hardware. At a fine-grained level, they execute assembly instructions in the form of machine code. Instructions have possibility of interfering with one another, causing the processor take more clock cycles to complete an instruction. The average amount of cycles a processor takes to complete an instruction is a metric called Cycles Per Instruction (CPI). A processor achieving a low CPI is able to get more work done in the same window of time as another processor. As CPI is a proxy for how efficiently a processor is carrying out work, correlating CPI to the amount of power consumed by a workload may open the door for a new metric that can be used to improve energy consumption. For example, if a scheduler can be designed to target a CPI number on each node in a heterogeneous cluster that yields an efficient return on the power consumed, such a scheduler would be able reduce energy consumption while maintaining, or perhaps even reducing, makespan.

6.2.4 Fine-grained Extrema

There are many ways of dynamically power capping a cluster. Extrema and Progressive Extrema both approach the problem from a global perspective. That is, both Extrema and Progressive Extrema take into account the historical power consumption of the cluster as a whole when making decisions. This opens the door for further work on strategies that focus on the local perspective. This may involve tracking the power consumption of each node individually and making decisions based upon that information.

6.2.5 Considering Streaming Workloads

The experiments in this thesis only account for tasks that finish. That is, they focus on batch workloads that have a beginning and end. Services, processes that must continuously run, may require a different set of strategies to mange co-incident power peaks and energy consumption. Examples of popular streaming platforms include Apache Storm [29], Apache Kafka [66], Apache Spark [75], Twitter's Heron [36], and LinkedIn's Samza [54]. This is due in large part to the ephemeral nature of the work they need to perform. Some services, such as databases and key-value storage, experience a variable amount of work to carry out over time. Thus, more investigation must be done to determine if the techniques presented here are applicable to such workloads.

Bibliography

[1] Tarek Abdelzaher and Mindi Yuan. "TAPA: Temperature aware power allocation in data center with Map-Reduce". In: *2011 International Green Computing Conference and Workshops*. IEEE, July 2011, pp. 1–8.

[2] Abed Abu-Dbai et al. "Enterprise Resource Management in Mesos Clusters". In: *Proceedings of the 9th ACM International on Systems and Storage Conference - SYSTOR '16*. 2016, pp. 1–1. ISBN: 9781450343817. DOI: `10.1145/2928275.2933272`. URL: `http://dl.acm.org/citation.cfm?doid=2928275.2933272`.

[3] Kristen Accardi. *Balancing Power and Performance in the Linux Kernel*. 2015. URL: `https://events.static.linuxfound.org/sites/events/files/slides/LinuxConEurope_2015.pdf`.

[4] Apache. *Apache Aurora*. 2017. URL: `http://aurora.apache.org/`.

[5] Arka A. Bhattacharya et al. "The need for speed and stability in data center power capping". In: *2012 International Green Computing Conference (IGCC)*. IEEE, June 2012, pp. 1–10. ISBN: 978-1-4673-2154-9. DOI: `10.1109/IGCC.2012.6322253`.

[6] Stephen M. Blackburn et al. “The DaCapo benchmarks”. In: *ACM SIGPLAN Notices* 41.10 (Oct. 2006), p. 169.

[7] Deva Bodas et al. “Simple Power-Aware Scheduler to Limit Power Consumption by HPC System within a Budget”. In: *2014 Energy Efficient Supercomputing Workshop*. IEEE, Nov. 2014, pp. 21–30. ISBN: 978-1-4799-7036-0.

[8] Brendan Burns et al. “Borg, Omega, and Kubernetes”. In: *Queue* 14.1 (2016), pp. 70–93. ISSN: 15427730. DOI: `10.1145/2898442.2898444`. URL: `http://dl.acm.org/citation.cfm?doid=2898442.2898444`.

[9] *Chameleon: A Large-scale, Reconfigurable Experimental Environment for Cloud Research*. 2014. URL: `https://www.chameleoncloud.org`.

[10] Howard David et al. “RAPL”. In: *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design - ISLPED ’10*. New York, New York, USA: ACM Press, 2010, p. 189.

[11] Pierre Delforge and Josh Whitney. “Data Center Efficiency Assessment”. In: (2014).

[12] Christina Delimitrou and Christos Kozyrakis. “IBench: Quantifying interference for datacenter applications”. In: *Proceedings - 2013 IEEE International Symposium on Workload Characterization, IISWC 2013*. 2013, pp. 23–33. ISBN: 9781479905539. DOI: `10.1109/IISWC.2013.6704667`.

[13] Christina Delimitrou et al. “Quasar: resource-efficient and QoS-aware cluster management”. In: *ACM SIGPLAN Notices* 49.4 (2014), pp. 127–144. ISSN: 0362-1340. DOI: `10.1145/2644865.2541941`.

[14] Renan DelValle et al. "Electron: Towards Efficient Resource Management on Heterogeneous Clusters with Apache Mesos". In: *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, June 2017, pp. 262–269. ISBN: 978-1-5386-1993-3. DOI: 10.1109/CLOUD.2017.41.

[15] Renan DelValle et al. "Exploiting Efficiency Opportunities Based on Workloads with Electron on Heterogeneous Clusters". In: *Proceedings of the10th International Conference on Utility and Cloud Computing*. UCC '17. New York, NY, USA: ACM, 2017, pp. 67–77. ISBN: 978-1-4503-5149-2. DOI: 10.1145/3147213.3147226. URL: http://doi.acm.org/10.1145/3147213.3147226.

[16] Renan DelValle et al. "Exploring the Design Space for Optimizations with Apache Aurora and Mesos". In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, June 2016, pp. 537–544. ISBN: 978-1-5090-2619-7. DOI: 10.1109/CLOUD.2016.0077.

[17] Emerson White Paper. "Energy Logic: Calculating and Prioritizing Your Data Center IT Efficiency Actions".

[18] Zacharia Fadika et al. "MARLA: MapReduce for Heterogeneous Clusters". In: *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE, May 2012, pp. 49–56. ISBN: 978-1-4673-1395-7. DOI: 10.1109/CCGrid.2012.135.

[19] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. "Power provisioning for a warehouse-sized computer". In: *ACM SIGARCH Computer Architecture News* (2007). ISSN: 01635964. DOI: 10.1145/1273440.1250665.

[20] Wes Felter et al. “An updated performance comparison of virtual machines and Linux containers”. In: *ISPASS 2015 - IEEE International Symposium on Performance Analysis of Systems and Software*. 2015, pp. 171–172. ISBN: 9781479919567. DOI: `10.1109/ISPASS.2015.7095802`.

[21] Jeffrey M Galloway, Karl L Smith, and Susan S Vrbsky. “Power Aware Load Balancing for Cloud Computing”. In: 2193.1 (2011). ISSN: 2078-0958.

[22] Ali Ghodsi et al. “Dominant Resource Fairness : Fair Allocation of Multiple Resource Types Maps Reduces”. In: *Ratio* (2011).

[23] Skand Gupta. *Mesos at Bloomberg*. URL: `https://www.youtube.com/watch?v=6w8PXklrC5I`.

[24] Jessica Hartog et al. “Configuring a MapReduce Framework for Dynamic and Efficient Energy Adaptation”. In: *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, June 2012, pp. 914–921. ISBN: 978-1-4673-2892-0. DOI: `10.1109/CLOUD.2012.137`.

[25] B Hindman, A Konwinski, and M Zaharia. “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.” In: *NSDI* (2011).

[26] Shengsheng Huang et al. “The HiBench benchmark suite: Characterization of the MapReduce-based data analysis”. In: *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. IEEE, 2010, pp. 41–51. ISBN: 978-1-4244-6522-4.

[27] Patrick Hunt et al. “ZooKeeper: Wait-free Coordination for Internet-scale Systems”. In: *USENIX Annual Technical Conference* 8 (2010), pp. 11–11. ISSN:

0210945x. URL: http://portal.acm.org/citation.cfm?id=1855851%5Cnhttps://www.usenix.org/event/usenix10/tech/full_papers/Hunt.pdf.

[28] Yuichi Inadomi et al. “Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC ’15*. 2015. ISBN: 9781450337236. DOI: 10.1145/2807591.2807638.

[29] Muhammad Hussain Iqbal and Tariq Rahim Soomro. “Big data analysis: Apache storm perspective”. In: *International journal of computer trends and technology* 19.1 (2015), pp. 9–14. URL: https://www.researchgate.net/profile/Tariq_Soomro/publication/271196175_Big_Data_Analysis_Apache_Storm_Perspective/links/54bff9900cf28a6324a02e6b.pdf.

[30] M Jette and M Grondona. “SLURM: Simple Linux Utility for Resource Management”. In: *ClusterWorld Conference and Expo CWCE*. Vol. 2682. 2003, pp. 44–60. ISBN: 978-3-540-20405-3. DOI: 10.1007/10968987. URL: http://www.springerlink.com/content/c4pgx63utdajtuwn/.

[31] John D. McCalpin. “Memory Bandwidth and Machine Balance in Current High Performance Computers”. In: *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (1995), pp. 19–25.

[32] Flavio P. Junqueira, Benjamin C. Reed, and Marco Serafini. “Zab: High-performance broadcast for primary-backup systems”. In: *Proceedings of the International*

Conference on Dependable Systems and Networks. 2011, pp. 245–256. ISBN: 9781424492336. DOI: 10.1109/DSN.2011.5958223.

[33] Cengiz Karakoyunlu and John A. Chandy. “Exploiting user metadata for energy-aware node allocation in a cloud storage system”. In: *Journal of Computer and System Sciences* 82.2 (Mar. 2016), pp. 282–309.

[34] Rini T. Kaushik and Milind Bhandarkar. “GreenHDFS: towards an energy-conserving, storage-efficient, hybrid Hadoop compute cluster”. In: (Oct. 2010), pp. 1–9.

[35] Kashif Nizam Khan et al. “How much power does your server consume? Estimating wall socket power using RAPL measurements”. In: *Computer Science - Research and Development* 31.4 (Nov. 2016), pp. 207–214. DOI: 10.1007/s00450-016-0325-4.

[36] Sanjeev Kulkarni et al. “Twitter Heron”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD ’15*. New York, New York, USA: ACM Press, May 2015, pp. 239–250. ISBN: 9781450327589. DOI: 10.1145/2723372.2742788.

[37] Leslie Lamport. “Paxos Made Simple”. In: *ACM SIGACT News* 32.4 (2001), pp. 51–58. ISSN: 01635700. DOI: 10.1145/568425.568433.

[38] Willis Lang and Jignesh M. Patel. “Energy management for MapReduce clusters”. In: *Proceedings of the VLDB Endowment* 3.1-2 (Sept. 2010), pp. 129–139.

[39] Andrew Leung, Andrew Spyker, and Tim Bozarth. "Titus: Introducing Containers to the Netflix Cloud". In: *Commun. ACM* 61.2 (Jan. 2018), pp. 38–45. ISSN: 0001-0782. DOI: 10.1145/3152529. URL: http://doi.acm.org/10.1145/3152529.

[40] Jacob Leverich and Christos Kozyrakis. "On the energy (in)efficiency of Hadoop clusters". In: *ACM SIGOPS Operating Systems Review* 44.1 (Mar. 2010), p. 61.

[41] Yu Li, Yi Liu, and Depei Qian. "A Heuristic Energy-aware Scheduling Algorithm for Heterogeneous Clusters". In: *2009 15th International Conference on Parallel and Distributed Systems*. IEEE, 2009, pp. 407–413. ISBN: 978-1-4244-5788-5. DOI: 10.1109/ICPADS.2009.33.

[42] Zhenhua Liu et al. "Data center demand response: Avoiding the coincident peak via workload shifting and local generation". In: *Performance Evaluation* 70.10 (2013), pp. 770–791.

[43] Muthucumaru Maheswaran et al. "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems". In: *Proceedings of the Eighth Heterogeneous Computing Workshop*. IEEE Computer Society Press, 1999, p. 232. ISBN: 0769501079.

[44] *Marathon: A container orchestration platform for Mesos and DC/OS*. URL: https://mesosphere.github.io/marathon/.

[45] Jason Mars et al. "Bubble-Up". In: *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 '11*. 2011. ISBN: 9781450310536. DOI: 10.1145/2155620.2155650.

[46] Silvano Martello, David Pisinger, and Daniele Vigo. "The Three-Dimensional Bin Packing Problem". In: *Operations Research* 48.2 (Apr. 2000), pp. 256–267.

[47] David Meisner et al. "Power management of online data-intensive services". In: *ACM SIGARCH Computer Architecture News* 39.3 (July 2011), p. 319.

[48] Merkel and Dirk. "Docker: lightweight Linux containers for consistent development and deployment". In: *Linux Journal* 2014.239 (2014), p. 2. ISSN: 1075-3583.

[49] Michael A Heroux et al. *Improving performance via mini-applications.* Tech. rep. Sandia National Laboratories, 2009.

[50] Justin Moore et al. "Making Scheduling " Cool ": Temperature-Aware Workload Placement in Data Centers". In: *Science* (2005). ISSN: 08832927. DOI: `10.1016/j.apgeochem.2009.03.006`.

[51] Nitin Naik. "Building a virtual system of systems using docker swarm in multiple clouds". In: *ISSE 2016 - 2016 International Symposium on Systems Engineering - Proceedings Papers.* 2016. ISBN: 9781509007936. DOI: `10.1109/SysEng.2016.7753148`.

[52] Ripal Nathuji and Karsten Schwan. "VirtualPower: coordinated power management in virtualized enterprise systems". In: *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles - SOSP '07* (2007). ISSN: 978-1-59593-591-5. DOI: `10.1145/1294261.1294287`.

[53] *NeRSC Benchmark Distribution and Run Rules.* 2016. URL: `http://www.nersc.gov/research-and-development/apex/apex-benchmarks/`.

[54] Shadi A Noghabi et al. “Samza: stateful scalable stream processing at LinkedIn”. In: *Proceedings of the VLDB Endowment* 10.12 (2017), pp. 1634–1645. URL: http://www.vldb.org/pvldb/vol10/p1634-noghabi.pdf.

[55] *Performance Co-Pilot*. 2016. URL: http://www.pcp.io/.

[56] Pavlos Petoumenos et al. “Power Capping: What Works, What Does Not”. In: *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, Dec. 2015, pp. 525–534. ISBN: 978-0-7695-5785-4. DOI: 10.1109/ICPADS.2015.72.

[57] *Phoronix Test Suite - Linux Testing and Benchmarking Platform, Automated Testing, Open-Source Benchmarking*. 2017. URL: http://www.phoronix-test-suite.com/.

[58] *Power Capping Framework*. 2016. URL: https://www.kernel.org/doc/Documentation/power/powercap/powercap.txt.

[59] Parthasarathy Ranganathan et al. “Ensemble-level power management for dense blade servers”. In: *Proceedings - International Symposium on Computer Architecture*. 2006. ISBN: 076952608X. DOI: 10.1109/ISCA.2006.20.

[60] Efraim Rotem et al. “Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge”. In: *IEEE Micro* 32.2 (Mar. 2012), pp. 20–27. ISSN: 0272-1732. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6148200.

[61] *Running Average Power Limit – RAPL*. 2014. URL: https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl.

[62] Sandia National Laboratories. *DGEMM*. 2016. URL: `http://www.nersc.gov/research-and-development/apex/apex-benchmarks/dgemm/`.

[63] Osman Sarood et al. “Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget”. In: *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Nov. 2014, pp. 807–818. ISBN: 978-1-4799-5500-8.

[64] *Scaling Mesos at Apple, Bloomberg, Netflix and more - Mesosphere*. 2015. URL: `https://mesosphere.com/blog/2015/08/25/scaling-mesos-at-apple-bloomberg-netflix-and-more/`.

[65] Manivannan Selvaraj and Soma Shekar Oruganti. *Paradigm Shift in CI at PayPal with Docker and Mesos*. URL: `https://www.youtube.com/watch?v=sC0A-Su_SP0`.

[66] Abhishek Sharma. *Apache kafka: Next generation distributed messaging system.* 2015.

[67] Craig A. Stewart et al. “Jetstream”. In: *Proceedings of the XSEDE16 on Diversity, Big Data, and Science at Scale - XSEDE16*. New York, New York, USA: ACM Press, 2016, pp. 1–8. ISBN: 9781450347556.

[68] “Tirias-Research-Whitepape”. 2014. URL: `https://www.amd.com/Documents/Tirias-Research-Whitepaper.pdf`.

[69] Pitt Turner et al. “Tier Classifications Define Site Infrastructure Performance”. In: *Uptime Institute* (2008).

[70] Arunchandar Vasan et al. "Worth their watts? - an empirical study of datacenter servers". In: *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture.* 2010. ISBN: 978-1-4244-5658-1. DOI: 10.1109/HPCA.2010.5463056.

[71] Vinod Kumar Vavilapalli et al. "Apache Hadoop YARN". In: *Proceedings of the 4th annual Symposium on Cloud Computing - SOCC '13.* New York, New York, USA: ACM Press, 2013, pp. 1–16. ISBN: 9781450324281.

[72] Matt Weinberger. *How Apple got Siri to run much faster, for a lot less cash.* URL: http://www.businessinsider.com/apple-siri-uses-apache-mesos-2015-8.

[73] Q Wu et al. "Dynamo: Facebook's Data Center-Wide Power Management System". In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA).* 2016, pp. 469–480. ISBN: 1063-6897. DOI: 10.1109/ISCA.2016.48.

[74] Xu Yang et al. "Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '13.* New York, New York, USA: ACM Press, 2013, pp. 1–11. ISBN: 9781450323789. DOI: 10.1145/2503210.2503264.

[75] Matei Zaharia et al. "Apache Spark: a unified engine for big data processing". In: *Communications of the ACM* 59.11 (2016), pp. 56–65. ISSN: 00010782. DOI: 10.1145/2934664.