Electron: Towards Efficient Resource Management on Heterogeneous Clusters with Apache Mesos

Renan DelValle, Pradyumna Kaushik, Abhishek Jain, Jessica Hartog, and Madhusudhan Govindaraju

Cloud and Big Data Lab, State University of New York (SUNY) at Binghamton
{rdelval1, pkaushi1, ajain13, jhartog1, mgovinda}@binghamton.edu

***Abstract*—As data centers continue to grow in scale, the resource management software needs to work closely with the hardware infrastructure to provide high utilization, performance, fault tolerance, and high availability. Apache Mesos has emerged as a leader in this space, providing an abstraction over the entire cluster, data center, or cloud to present a uniform view of all the resources. In addition, frameworks built on Mesos such as Apache Aurora, developed within Twitter and later contributed to the Apache Software Foundation, allow massive job submissions with heterogeneous resource requirements. The availability of such tools in the Open Source space, with proven record of large-scale production use, make them suitable for research on how they can be adapted for use in campus-clusters and emerging cloud infrastructures for different workloads in both academia and industry. As data centers run these workloads and strive to maintain high utilization of their components, they suffer a significant cost in terms of energy and power consumption. To address this cost we have developed our own framework, Electron, for use with Mesos. Electron is designed to be configurable with heuristic-driven power capping policies along with different scheduling policies such as Bin Packing and First Fit. We characterize the performance of Electron, in comparison with the widely used Aurora framework. On average, our experiments show that Electron can reduce the 95th percentile of CPU and DRAM power usage by 27.89%, total energy consumption by 19.15%, average power consumption by 27.90%, and max peak power usage by 16.91%, while maintaining a similar makespan when compared to Aurora using the proper combination of power capping and scheduling policies.**

I. Introduction

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 [1] and Chameleon [2]. Among the software management tools being deployed in data centers and clouds, Apache Mesos [3], 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 [4]. Thus, our focus is on reducing peak power draws and energy consumption without having a large impact on makespan and resource utilization.

Marathon [5] and Aurora [6] 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 [7] 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 [8]. 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.

II. Mesos, Aurora, & Electron

A. Mesos

Mesos has evolved from a proof of concept to a widely utilized software capable of providing scalability and fault tolerance to massive scale applications. Mesos is currently deployed in several cloud infrastructures across the industry and its use cases include Apple's Siri, Bloomberg's data analytics, and Paypal's continuous integration system [9].

Apache Mesos provides a layer of abstraction over the bare metal in data centers and large clusters. It combines cluster resources (CPU, memory, and storage) into a shared pool that is then distributed to competing applications based on their fine-grained resource needs [3]. Applications that run on Mesos, called *frameworks*, view the Mesos layer as a cluster-wide, highly-available, fault-tolerant, distributed operating system. In our experiments, we use Mesos as an abstraction layer and only run a single framework at a time.

Mesos works as follows:

- A *worker* daemon, known as a Mesos Agent, analyzes the machine on which it runs, discovers available resources, and advertises them cluster-wide through the Mesos Master.
- The Mesos Master then makes these resources available to registered frameworks by sending coarse-grained *Resource Offers* for each node in the cluster. Frameworks may refuse an offer if it does not suit their needs. Resources are returned to the Mesos pool when (a) the framework refuses an offer, (b) a task completes or fails, thereby freeing up host resources, (c) Mesos rescinds an offer, or (d) only a portion of the offer is accepted.

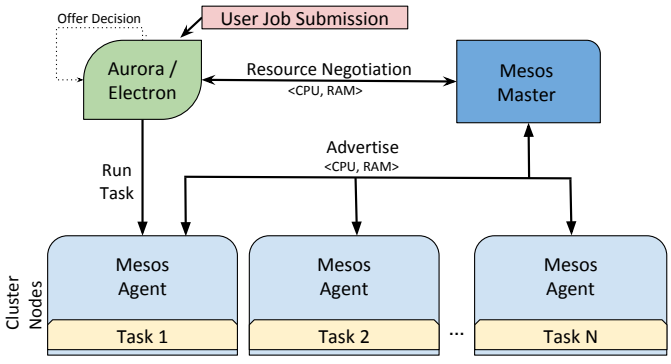


Fig. 1. Architecture of Mesos and Aurora/Electron. Note that Electron has the same position in the architecture as Aurora.

B. Apache Aurora

Aurora allows jobs to be run on Mesos managed resources, as shown in Figure 1. Aurora has its own Domain Specific Language that enables users to create a Job configuration. A Job consists of a collection of Tasks, each of which is comprised of processes, which are understood and managed either by the Thermos executor, bundled with Apache Aurora, or by Mesos supported containers. Jobs fall into three types: services, cron, or ad-hoc. Service type jobs are run continuously, cron type jobs are run at intervals, and ad-hoc type jobs are only run once. For our experiments, we chose ad-hoc type jobs as we needed workloads to complete in order to quantify makespan and how much power was used for a standardized workload.

C. Electron

Electron meets the need for a more fine-grained control in order to make informed choices regarding resources offered by Mesos. As shown in Figure 1, Electron is used in the same way as Aurora, acting as a light-weight framework capable of scheduling tasks in Docker containers on Mesos. However, in addition to being a scheduler, Electron also takes advantage of tools such as Performance Co-Pilot [10] and RAPL [11] to help contain the power envelope within defined thresholds, reduce peak power consumption, and also reduce total energy consumption. Electron is able to leverage the Mesos-provided resource abstraction to allow different algorithms to decide how to consume resource offers made by a Mesos Master. This abstraction layer has been shown to effectively scale when used in very large clusters [3] [12].

D. Differences Between Electron and Aurora

There are several differences between Electron and Aurora:

- Aurora is heavy-weight by design as it is built to support execution of tasks, services, and cron jobs.
- As a production level software, Aurora stores multiple statistics that are kept for analysis by DevOps.
- Unlike Aurora, Electron does not currently support the preemption of tasks.
- Aurora has chosen First Fit as its only scheduling policy due to the fact that it makes supporting more complex features, such as preemption, easier.
- Electron can be configured to use different scheduling policies in conjunction with different power capping strategies.

It should be noted that no feature exclusive to Aurora was used in our experiments. Both Electron and Aurora were run on an external node that was not monitored for performance due to the assumption that key resource management modules typically do not run on worker nodes. Furthermore, this work focuses on the execution of the workload, not on the performance of the Mesos frameworks themselves that are responsible for scheduling the workload.

III. 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. While these are well-known algorithms, to the best of our knowledge, their impact on power and energy consumption in Mesos environment has not been studied.

A. Electron First Fit (Electron FF)

The First Fit algorithm seeks to find an offer that satisfies the resource constraints of a task waiting to be scheduled. If an offer with sufficient resources is found, the offer is consumed by using it to schedule the task on the Mesos cluster. If none of the offers contain sufficient resources to schedule the task, that task will remain in the queue until the next round of Mesos offers is received. The order in which tasks are scheduled is left to the framework designer. For these experiments, we chose a First Come First Served (FCFS) queue due to its similarity to Aurora's queue design.

B. Electron Bin Packing (Electron BP)

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.

IV. 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 [13], while Petoumenos et al. have quantified RAPL's ability to cap power consumption successfully [14]. 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.

[1]Sandy Bridge architecture was released in 2011.

A. 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 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 static capping 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 (described in section V) 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.

B. 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 1, 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. A cluster wide Average Historical Power ($AHP_{cluster}$) value is maintained and compared against a *high threshold* and a *low threshold*. If $AHP_{cluster}$ exceeds the *high threshold*, the node with the highest Individual Average Historical Power (AHP_{node}) value is capped. Otherwise, if $AHP_{cluster}$ is below the *low threshold*, the most recently capped node 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.

V. 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
- 2 *Class B* nodes - Two 8 core, 16 thread Intel Xeon E5-2640 v3 @ 2.60GHz and 64 GB RAM

Algorithm 1 Extrema Dynamic Capping algorithm

```
procedure EXTREMA_CAP(Threshold)
    AHP_cluster ← Avg(ClusterPower)
    CappedNodes ← Stack()
    if AHP_cluster > Threshold.Hi then
        Victims ← Sort_non−inc(AHP_node)
        for Victim in Victims do
            if Victim not in CappedNodes then
                Cap(Victim)
                CappedNodes.Push(Victim)
                break
            end if
        end for
    end if
    if AHP_cluster < Threshold.Low then
        Uncap(CappedNodes.Pop())
    end if
end procedure
```

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 [15]	Finite element generation, assembly and solution for an unstructured grid problem.	CPU
DGEMM [16]	Multi-threaded, dense-matrix multiplication.	CPU
STREAM [17]	Calculates sustainable memory bandwidth and the computation rate for simple vector kernels.	Memory

Table V. *Workload benchmarks*

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

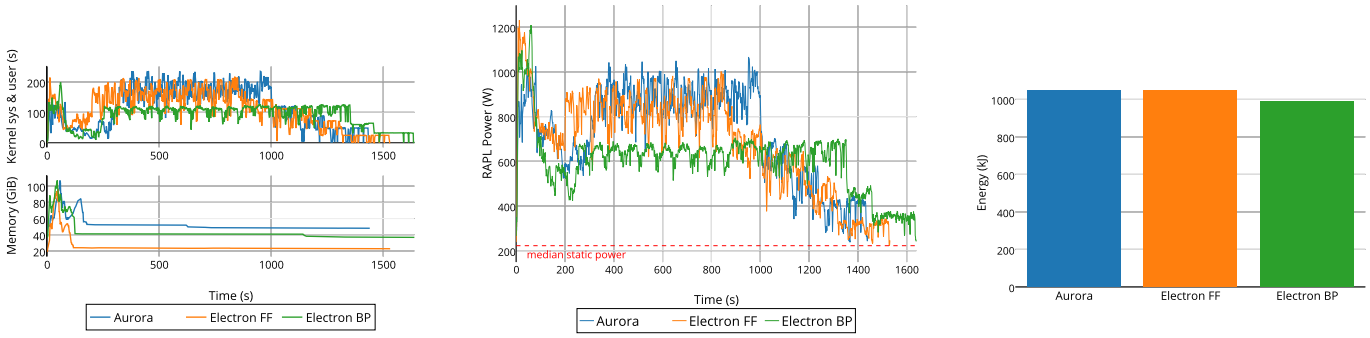
The classification of the nodes into power classes in the Stratos cluster was done based on the potential power consumption by processors on each of those nodes. Class A nodes have the highest Thermal Design Power (TDP), followed by Class B, and finally Class C. The higher the TDP, the more Watts a processor 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 [10] 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. For our static capping experiments, an Ansible playbook is used to statically cap all nodes in the cluster using the Linux Power Capping Framework [18]. 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 [19], as listed in Table V. The workloads with which we assess performance were derived from the DaCapo Benchmark suite, Phoronix Benchmark suite [20], MiniFE from Mantevo [15], as well as Stream and Dgemm from NERSC [21]. 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.

VI. Performance Analysis

A. Aurora vs. Electron First Fit (Electron FF)

In Figure 2, we can observe that Aurora and Electron, with a First Fit policy, exhibit similar performance. The similarity in performance is backed by a 1.19% difference in the 95^{th} percentile for power consumption. The difference in the mean power consumption, however, is lower by 6.09% 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 2b. 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 5.74% in the mean CPU Time used to finish all computations by 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 51.12% lower. We speculate that this difference in memory usage may be due to



(a) Cluster-wide CPU & Memory Utilization (b) Cluster-wide Power Consumption (c) Cluster-wide Energy Consumption

Fig. 2. A comparison between Aurora and both of Electron's scheduling policies without the use of a power cap.

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 2a between Aurora and Electron FF.

B. Electron FF vs. Electron Bin Packing (Electron BP)

A comparison between Electron FF and Electron Bin Packing can also be seen in Figure 2. Electron BP suffers an increase in makespan compared to Electron FF of 7.05% and 13.81% compared to Aurora. However, the 95^{th} percentile of power consumption for the Electron BP run is decreased by 27.02% compared to Electron FF and a similar percentage when compared to Aurora. Overall energy consumption is decreased by 5.89% compared to Electron FF and decreased by 6.04% compared to Aurora. The mean CPU Time needed to finish all workloads by Electron BP is 18.96% lower than Electron FF and 23.39% lower than Aurora. Mean memory consumption is 19.14% lower compared to Aurora.

C. Effects of Static Power Capping

1) Aurora-S: Setting a static cap for the entire cluster had a beneficial impact peak power and energy usage when compared to the uncapped run. Aurora-S experienced a 15.68% reduction in the 95^{th} percentile of power across the cluster. The max peak power was reduced by 16.91% while overall energy consumption was reduced by 4.34%. However, static capping also resulted in a 14.02% increase in makespan. Mean CPU time decreased by 3.73% while memory utilization increased by 7.25%.

2) Electron FF-S: Electron FF run under static capping (Electron FF-S) experienced a 14.09% reduction in the 95^{th} percentile of power and a 24.53% reduction of the max peak power draw when compared to its uncapped run. Electron FF-S experienced a makespan increase of 2.48%. Overall energy was reduced by 5.10%. Mean CPU time increased by 12.26%. 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 clock ticks taking place in a second is reduced, and so is the amount of work a core is able to do. Because less work is being done per unit of time, it takes the CPU longer to perform the same work done by the uncapped run.

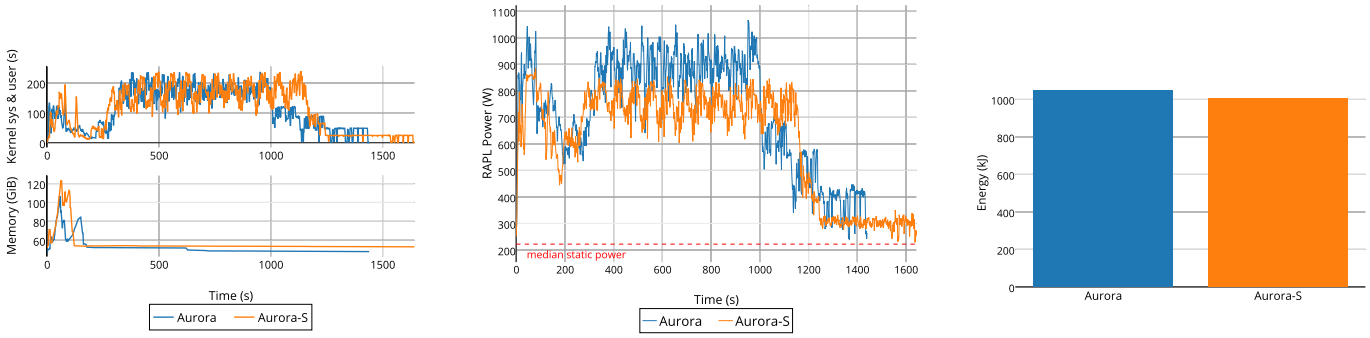
3) Electron BP-S: Electron BP-S finished all workloads with an increase in makespan of 7.74% in comparison to the uncapped run. The power in the 95^{th} percentile increases as well by 3.86%. However, the max peak power experienced a decrease of 21.59% and overall energy was reduced by 4.74%. Mean CPU Time increased 8.81% while mean memory usage increased by 23.90%.

Taking into consideration all statically capped runs, two patterns can be observed: (a) makespan experiences a slight to drastic increase and (b) max peak power consumption is severely reduced. Both effects are the result of setting a loose upper bound before which the cluster will throttle itself in an attempt to meet the power constraint. Since this loose upper bound exists throughout the execution, statically capping is better suited in guarding against sudden power spikes caused by the governor increasing the CPU frequencies.

D. Effects of Extrema Power Capping

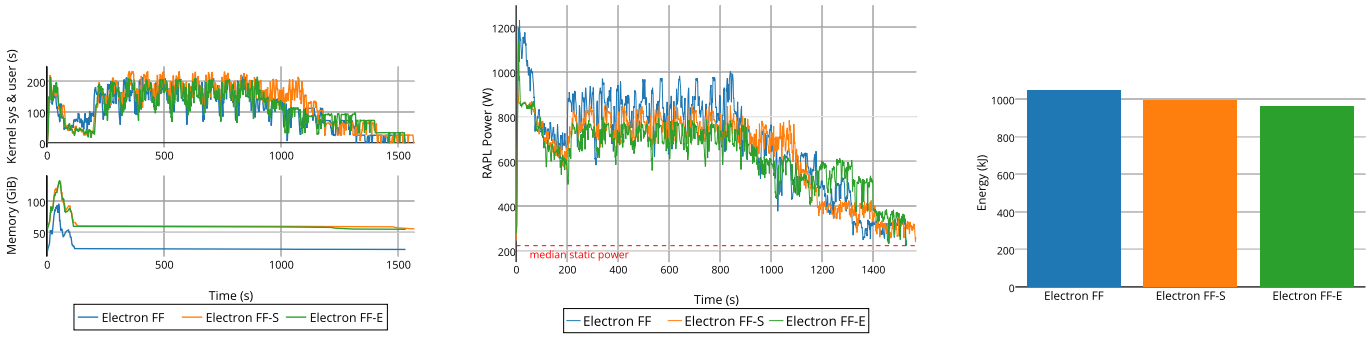
1) Electron FF-E: Electron with a First Fit policy running under Extrema capping (Electron FF-E) experiences a similar makespan to the uncapped run, differing by 0.07%. However, when compared to the static capped run, Electron FF-E experiences a decrease in makespan of 2.36%. The power readings in the 95^{th} percentile are improved in Electron FF-E by 18.97% and 5.69% when compared to Electron FF and Electron FF-S respectively. The max peak power of Electron FF-E decreased by 7.53% compared to the uncapped run while it fared worse against the static capped run with an increase of 22.53%. Energy consumption is lowered by 8.28% compared to Electron FF and 3.35% to Electron FF-S. Mean CPU time is 4.01% higher compared to the uncapped run and 7.79% lower compared to the statically capped run. Mean memory consumption increased by 137.17% compared to Electron FF but was only 2.36% lower than Electron FF-S leading us to believe that the memory consumption experienced by Electron FF is an outlier.

2) 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 1.46% difference in makespan between Electron BP and Electron BP-E. The 95^{th} percentile of power consumption for Electron BP-E is



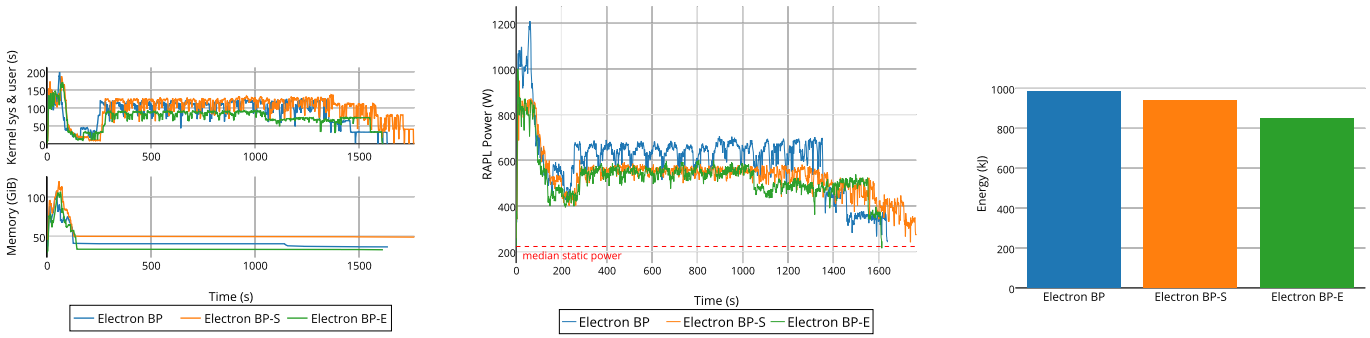
(a) Cluster-wide CPU & Memory Utilization (b) Cluster-wide Power Consumption (c) Cluster-wide Energy Consumption

Fig. 3. Comparison between uncapped Aurora and Aurora running under a static power cap.



(a) Cluster-wide CPU & Memory Utilization (b) Cluster-wide Power Consumption (c) Cluster-wide Energy Consumption

Fig. 4. Comparison of uncapped Electron First Fit against Electron First Fit with static capping, and Electron First Fit under Extrema capping.



(a) Cluster-wide CPU & Memory Utilization (b) Cluster-wide Power Consumption (c) Cluster-wide Energy Consumption

Fig. 5. Comparison of uncapped Electron Bin Packed, statically capped Electron Bin Packed and Electron Bin Packed running under Extrema capping.

higher by 2.33% than Electron BP and lower by 1.47% when compared Electron BP-S. The max peak power consumption is decreased compared to the uncapped run by 17.27%, while it resulted in an increase of 5.51% compared to the static capped run.

The difference in the max peak power draw between statically capped runs and the Extrema capped runs can be attributed to the Extrema capping algorithm being reactive to power trends. It is only after a sudden power spike is experienced that Extrema attempts to mitigate the high power draw. An example of this scenario occurs in our experiments when the initial batch of benchmarks are scheduled across the cluster. During this time period, power draw increases dramatically as each node's governor increases the CPU frequency to handle the incoming load. The spike, however, is short lived as Extrema is able to bring down power consumption to levels similar to those observed in the statically capped run after second ten.

VII. Related Work

In our previous work [22], we characterized the performance of Aurora and Mesos for various workloads to identify the opportunities available for optimization. We used a pre-scheduler

to assign benchmarks to run on specific nodes. In contrast, this paper introduces a Mesos framework, Electron, that (a) can dynamically choose to run a task on any node in the cluster instead of restricting a task to run on a predetermined node, and (b) combines scheduling policies and power capping strategies to limit the peak power consumption and the overall energy consumption. Additionally, we introduced the Extrema dynamic power capping policy that uses RAPL to dynamically cap the cluster at configurable thresholds. We are not aware of any other work that quantifies, compares, and contrasts the impact that different power capping strategies have on different policies for launching jobs on a Mesos cluster.

To this end, we discuss and compare the ideas in power and energy management that are related to the work presented here. We note that a lot of work in power and energy efficiency is complementary to our work, such as (a) microarchitecture level optimizations, (b) turning off or suspending nodes at the cluster level, and (c) using Dynamic-Voltage and Frequency-Scaling (DVFS) schemes.

Several efforts have been directed towards dividing a cluster into hot or cold zones [23] 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 [24]. Lang and Patel 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 [25].

Abdelzaher et al. [26] used CPU temperature for scheduling jobs on a MapReduce cluster. In our earlier work on MapReduce based frameworks [27], we also quantified the relationship between CPU temperature and energy consumption and adapted the MARLA MapReduce framework [28] 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. [29] 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. [30] developed a power aware scheduler for SLURM. 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, however, allows changing of processor frequency across the benchmarks that were run. Meanwhile, our approach 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.

In a similar approach to ours, Yang et al. [31] proposed 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 the jobs with higher power consumption during the off-peak period, and the jobs with lower power consumption during the on-peak period. While this approach works for batch jobs, our focus is on scheduling jobs that are currently in queue to be launched on Mesos. Our framework can be extended to incorporate other policies, such as those presented by Yang et al., for batch jobs.

Sarood et al. [32] proposed 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. In our work, we assume that the resources allocated to a job cannot be changed as that is a tenet of Mesos's current design.

VIII. 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 makespan. However, Extrema suffers in comparison to static capping for reducing max peak powers 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.

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 6.04% for the tested benchmarks. Additionally, the median power usage is reduced by 19.66%.

- Static capping works well when the power consumption of the benchmarks in the workload stays below or slightly above the static cap value. Furthermore, since it sets a loose upper bound that is always in place, it is better suited to deal with sudden power spikes.
- Compared to the uncapped run, the statically capped runs show a reduced power ceiling ranging from a decrease of 16.91% for Aurora to a decrease of 24.53% 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 19.14% reduction in energy consumption and 27.90% reduction in mean power consumption.

REFERENCES

[1] C. A. Stewart, D. C. Stanzione, J. Taylor, E. Skidmore, D. Y. Hancock, M. Vaughn, J. Fischer, T. Cockerill, L. Liming, N. Merchant, T. Miller, and J. M. Lowe, "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.

[2] "Chameleon: A Large-scale, Reconfigurable Experimental Environment for Cloud Research." [Online]. Available: https://www.chameleoncloud.org

[3] B. Hindman, A. Konwinski, and M. Zaharia, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." *NSDI*, 2011.

[4] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar, "The need for speed and stability in data center power capping," in *2012 International Green Computing Conference (IGCC)*. IEEE, 6 2012, pp. 1–10.

[5] "Marathon: A container orchestration platform for Mesos and DC/OS." [Online]. Available: https://mesosphere.github.io/marathon/

[6] Apache, "Apache Aurora," 2017. [Online]. Available: http://aurora.apache.org/

[7] Merkel and Dirk, "Docker: lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[8] S. Martello, D. Pisinger, and D. Vigo, "The Three-Dimensional Bin Packing Problem," *Operations Research*, vol. 48, no. 2, pp. 256–267, 4 2000.

[9] "Scaling Mesos at Apple, Bloomberg, Netflix and more - Mesosphere." [Online]. Available: https://mesosphere.com/blog/2015/08/25/scaling-mesos-at-apple-bloomberg-netflix-and-more/

[10] "Performance Co-Pilot." [Online]. Available: http://www.pcp.io/

[11] "Running Average Power Limit RAPL." [Online]. Available: https://01.org/blogs/2014/running-average-power-limit--rapl

[12] "4 Unique Ways Uber, Twitter, PayPal, and Hubspot Use Apache Mesos." [Online]. Available: https://www.linux.com/news/4-unique-ways-uber-twitter-paypal-and-hubspot-use-apache-mesos

[13] K. N. Khan, Z. Ou, M. Hirki, J. K. Nurminen, and T. Niemi, "How much power does your server consume? Estimating wall socket power using RAPL measurements," *Computer Science - Research and Development*, vol. 31, no. 4, pp. 207–214, 11 2016.

[14] P. Petoumenos, L. Mukhanov, Zheng Wang, H. Leather, and D. S. Nikolopoulos, "Power Capping: What Works, What Does Not," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 12 2015, pp. 525–534.

[15] Michael A Heroux, Douglas W Doerfler, Paul S Crozier, James M Willenbring, H Carter Edwards, Alan Williams, Mahesh Rajan, Eric R Keiter, Heidi K Thornquist, and Robert W Numrich, "Improving performance via mini-applications," Sandia National Laboratories, Tech. Rep., 2009.

[16] Sandia National Laboratories, "DGEMM," 2016. [Online]. Available: http://www.nersc.gov/research-and-development/apex/apex-benchmarks/dgemm/

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

[18] "Power Capping Framework." [Online]. Available: https://www.kernel.org/doc/Documentation/power/powercap/powercap.txt

[19] S. M. Blackburn, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. Eliot, B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, R. Garner, D. von Dincklage, B. Wiedermann, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, and D. Frampton, "The DaCapo benchmarks," *ACM SIGPLAN Notices*, vol. 41, no. 10, p. 169, 10 2006.

[20] "Phoronix Test Suite - Linux Testing and Benchmarking Platform, Automated Testing, Open-Source Benchmarking." [Online]. Available: http://www.phoronix-test-suite.com/

[21] "NeRSC Benchmark Distribution and Run Rules." [Online]. Available: http://www.nersc.gov/research-and-development/apex/apex-benchmarks/

[22] Renan DelValle, Gourav Rattihalli, Angel Beltre, Madhusudhan Govindaraju, and Michael Lewis, "Exploring the Design Space for Optimizations with Apache Aurora and Mesos," in *IEEE International Conference on Cloud Computing (CLOUD), Applications Track, 2016*, 2016.

[23] R. T. Kaushik and M. Bhandarkar, "GreenHDFS: towards an energy-conserving, storage-efficient, hybrid Hadoop compute cluster," pp. 1–9, 10 2010.

[24] J. Leverich and C. Kozyrakis, "On the energy (in)efficiency of Hadoop clusters," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, p. 61, 3 2010.

[25] W. Lang and J. M. Patel, "Energy management for MapReduce clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 129–139, 9 2010.

[26] T. Abdelzaher and M. Yuan, "TAPA: Temperature aware power allocation in data center with Map-Reduce," in *2011 International Green Computing Conference and Workshops*. IEEE, 7 2011, pp. 1–8.

[27] J. Hartog, Z. Fadika, E. Dede, and M. Govindaraju, "Configuring a MapReduce Framework for Dynamic and Efficient Energy Adaptation," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 6 2012, pp. 914–921.

[28] Z. Fadika, E. Dede, J. Hartog, and M. Govindaraju, "MARLA: MapReduce for Heterogeneous Clusters," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE, 5 2012, pp. 49–56.

[29] C. Karakoyunlu and J. A. Chandy, "Exploiting user metadata for energy-aware node allocation in a cloud storage system," *Journal of Computer and System Sciences*, vol. 82, no. 2, pp. 282–309, 3 2016.

[30] D. Bodas, J. Song, M. Rajappa, and A. Hoffman, "Simple Power-Aware Scheduler to Limit Power Consumption by HPC System within a Budget," in *2014 Energy Efficient Supercomputing Workshop*. IEEE, 11 2014, pp. 21–30.

[31] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M. E. Papka, "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.

[32] O. Sarood, A. Langer, A. Gupta, and L. Kale, "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, 11 2014, pp. 807–818.