QoR-Aware Power Capping for Approximate Big Data Processing

Seyed Morteza Nabavinejad[†*], Xin Zhan[†], Reza Azimi[†], Maziar Goudarzi[*] and Sherief Reda[†]
[†] School of Engineering, Brown University, Rhode Island, USA
[*]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Abstract*—To limit the peak power consumption of a cluster, a centralized power capping system typically assigns power caps to the individual servers, which are then enforced using local capping controllers. Consequently, the performance and throughput of the servers are affected, and the runtime of jobs is extended as a result. We observe that servers in big data processing clusters often execute big data applications that have different tolerance for approximate results. To mitigate the impact of power capping, we propose a new power-Capping aware resource manager for Approximate Big data processing (CAB) that takes into consideration the minimum Quality-of-Result (QoR) of the jobs. We use industry-standard feedback power capping controllers to enforce a power cap quickly, while, simultaneously modifying the resource allocations to various jobs based on their progress rate, target minimum QoR, and the power cap such that the impact of capping on runtime is minimized. Based on the applied cap and the progress rates of jobs, CAB dynamically allocates the computing resources (i.e., number of cores and memory) to the jobs to mitigate the impact of capping on the finish time. We implement CAB in Hadoop-2.7.3 and evaluate its improvement over other methods on a state-of-the-art 28-core Xeon server. We demonstrate that CAB minimizes the impact of power capping on runtime by up to 39.4% while meeting the minimum QoR constraints.

I. Introduction

Power is an important shared resource for large-scale clusters and data centers [9], [4], [1]. During the past decade, a large body of work has been done on power management and peak power capping for clusters [11], [5], [10], [3]. Given the fact that servers generally operate at low utilization, power is usually over-subscribed in clusters to improve the cost efficiency. Power provisioning units connect more servers than their designed capacity. Power capping techniques aim to avoid the capacity violation in case the servers consume more power than the available capacity [10], [3]. Power capping system assigns individual power caps to the servers to prevent total power violation. As a result, the runtime of jobs hosted on capped servers can increase subsequently.

Companies and institutions maintain their private clusters for big data processing mainly due to preserve the data security. Private clusters process various types of big data jobs. Some of these jobs are periodically executed to extract information from data. These jobs are intrinsically approximate and approximation techniques can be applied to optimize their runtime. For example, Figure 1 shows the accuracy (QoR) of various approximation degrees for five selected big data jobs running on the Hadoop framework. Here we use the fraction of executed tasks of a job as the approximation knob. The QoR of jobs with different fraction of completed tasks are quantified by comparing the magnitude of partial results to the magnitude of final results when all tasks are completed. Some jobs quickly achieve decent QoR while only completing a small fraction of tasks. Many of the jobs in private clusters are recurrent and can be characterized based on historical data. System administrators can accordingly identify the QoR as a function of the fraction of executed tasks.

Considering the QoR variety illustrated in Figure 1, there is a potential opportunity to reallocate the computing resources among jobs to minimize the impact of capping on runtime at the expense of acceptable reduction in QoR. In this paper, We propose *CAB* for power-**C**apping aware resource manager for **A**pproximate **B**ig data processing. *CAB* is a dynamic resource management system that leverages approximation techniques to reduce the impact of power capping on runtime of big data jobs, while meeting QoR constraints. CAB is fully implemented in Hadoop-2.7.3. Our contributions are as follows.

- We devise a method to approximate big data jobs during runtime by dynamically adjusting their number of tasks. In conjunction, we devise a multiple queuing scheme to sort jobs with different *QoR sensitivities*, where *sensitivity* is defined as the fraction of tasks that are guaranteed to be executed to meet a minimum QoR. Jobs are then submitted to the appropriate queue based on their minimum QoR requirements.
- We propose a resource management system that optimizes the allocation of resources dynamically depending on the applied power caps. *CAB* incorporates a novel power-aware auto-regressive estimation filter that uses job progress rate, i.e., tasks completed per second, and applied power caps to estimate the expected finish time. *CAB* then uses an optimization methodology to reallocate the computing resources (i.e., CPU cores and memory) among the jobs, such that the impact of power capping on runtime is minimized and the QoR constraints are met within the power cap.
- We implement *CAB* in Hadoop-2.7.3 and deploy it on a 28-core Xeon server with an industry-standard power cap controller. We evaluate *CAB* using various Hadoop benchmarks and power caps. Comparing against previous works that

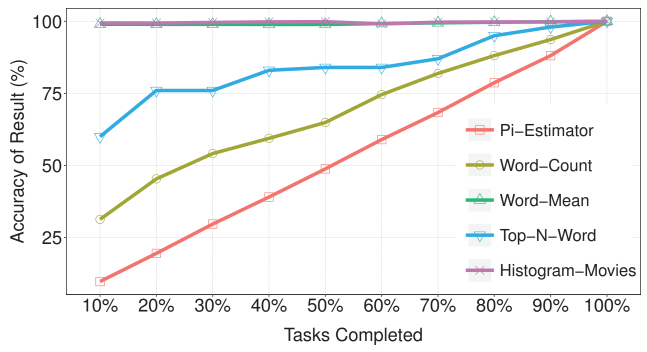

Fig. 1. The accuracy of partial results obtained for five big data jobs.

used approximation techniques (e.g., ApproxHadoop [6]), we show that *CAB* reduces the impact of power capping on runtime by up to 10.7%.

Compared to previous works that used approximation techniques to trade-off execution time at the expense of small amounts of inaccuracies [6], [7], [8], this work has important distinctions. Previous approaches considered a baseline static runtime; in our approach, the baseline runtime is a function of the applied power cap. Furthermore, we consider an environment where multiple jobs are running simultaneously, whereas prior works focused on trading-off accuracy and runtime of a single job [8]. Our approximation technique is applied dynamically depending on the applied power caps. The approximation degree is thus adaptive to the status of the system. Prior works consider static approximation techniques, where the approximation degree is decided statically at job launch time [6].

II. Methodology

Our methodology, *CAB*, is implemented on a Hadoop server, where computing resources are discretized in form of containers. Each container is configured with equal number of CPU cores and equal amount of memory. To sort jobs by their approximation tolerance, multiple job queues are created, as illustrated in Figure 2. Queues have different QoR sensitivities from high to low, where the *sensitivity* of a queue is the minimum fraction of tasks that is guaranteed to be completed for a job submitted to the queue. For example, a high-sensitivity queue with 90%-task guarantee will always complete at least 90% of all tasks from a job that is submitted to it. In addition to queues, *CAB* consists of three main components: *CAB-estimator*, *CAB-allocator* and *CAB-controller*. Given the input power cap and progress rates of the jobs in the system, *CAB-estimator* estimates the expected runtimes of the jobs. It then gives them to the *CAB-allocator* which allocates the resources, in form of containers, to the queues so as to minimize the impact of capping on runtime while meeting the minimum required QoR for the jobs. The allocation results together with the target number of tasks for each job are then enforced using the *CAB-controller*.

CAB-estimator. To estimate the finish time of jobs, we first define the *progress rate* of job i, v_i, as the number of completed tasks of job i per second. An applied power cap decreases the frequency and voltage of CPU and subsequently reduces the progress rate of jobs. For large big data processing systems, it is not practical to train prediction models for every job under various power caps. Additionally, multiple co-running jobs on the same server introduce resource contention, which can lead to discrepancies against *a priori* model. Instead, we propose a novel *power-aware and resource-aware auto-regressive filter* on the latest n progress rate samples to estimate runtime directly during execution.

At each sampling interval, *CAB-estimator* measures the progress rate for each job by dividing the number of completed tasks within an interval of time by the interval's period. Let $c_i[k]$ be the number of containers allocated to job i at time k and $p[k]$ be the power cap at time k. Given the past progress rates, $v_i[\cdot]$, the estimated progress rate $\hat{v}_i[k]$ of job i can be written as a function of resource allocation under the current power cap:

$$\hat{v}_i[k] = \sum_{j=1}^{n} \varphi_j \frac{p[k]}{p[k-j]} \frac{c_i[k]}{c_i[k-j]} v_i[k-j], \tag{1}$$

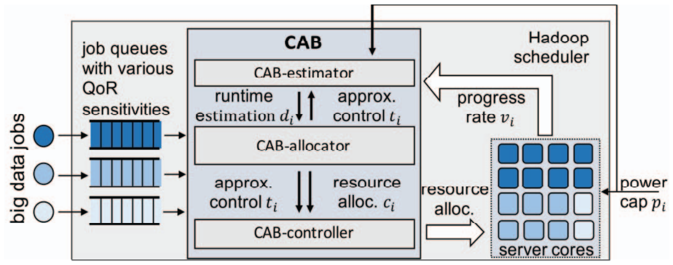

Fig. 2. An overview of *CAB*.

where φ denotes the weight for each of the previous n samples and $\sum_j \varphi_j = 1$. In the simplest case, $\varphi_j = 1/n$. Note the auto regressive filter scales the history of samples by the ratios of current and past power caps and resource allocations.

Let t_i^{min} indicate the minimum number of tasks that should be executed and $t_i[k]$ stand for number of executed tasks until time k. For each job i, the *finish time estimate*, $d_i[k]$, is equal to the current runtime of the job, $r_i[k]$, plus the number of remaining tasks to be completed, $t_i - t_i[k]$, divided by the estimated progress rate, $\hat{v}_i[k]$; i.e.,

$$d_i[k] \quad = r_i[k] + \frac{t_i^{min} - t_i[k]}{\hat{v}_i[k]}. \tag{2}$$

The estimate of the finish times for all jobs are continuously computed and sent to *CAB-allocator* for optimization.

CAB-allocator. In the absence of power caps, the finish time estimation of a job will be calculated by *CAB-estimator* and kept as the target runtime D_i if power caps are applied. When a power cap is applied, *CAB-allocator* will try to optimize the runtime of job i to match D_i by applying approximation and rearranging the container allocation to minimize the runtime stretch, while meeting the minimum QoR requirement of the corresponding job queues.

Our objective is to minimize the runtime stretch from the expected finish time target D_i of each job i, while adding no benefit for early completion. We define the *objective function* as the sum of the runtime increase of the jobs normalized to their expected remaining runtime, under the *constraint* that the minimum number of tasks that is guaranteed by the queue has to be completed. The proposed heuristic (see Figure 3) tries to allocate the resources to each job proportionally to the ratio of its estimated runtime stretch to total runtime stretch of all the jobs. Using this proportional allocation method, *CAB* assures that jobs that suffer from power cap more than others, receive more containers to conquer its detrimental impact on their runtime subject to QoR constraints. Hence, the resource share of each job can be calculated by (3).

$$c_i[k+1] = c^{max} \frac{d_i[k]/D_i}{\sum_i d_i[k]/D_i} \tag{3}$$

CAB-controller. The Hadoop MapReduce job method `computeProgress()` is accessed periodically by *CAB-controller* to evaluate the progress of each job. At each control epoch, only the jobs that have exceeded their expected runtime D_i are considered. For each of these jobs, if the number of completed tasks meet or exceed t_i^{min}, then *CAB-controller* drops the rest of the map tasks and returns the approximate results. If the minimum number of tasks has not been completed yet, the controller allows jobs to continue despite the runtime violation. *CAB-controller* also receives the containers allocation, c_i, for each job queue. To set the maximum resource share of each queue from total resources,

Procedure: *CAB-allocatora and CAB-controller algorithm*
Input: estimated finish time of each job $d_i[k]$,
total number of containers c^{max},
expected finish time of each job D_i,
approximation range of each job $[t_i^{min}, t_i^{max}]$
Output: container allocation $c_i[k+1]$

```
1. While $t_i[k] < t_i^{min}$ or $d_i[k]/D_i < 1$ :
2.     $c_i[k+1] = c^{max} \frac{d_i[k]/D_i}{\sum_i d_i[k]/D_i}$
3.     Update $d_i$ from CAB-estimator
4. If $d_i[k] \geq D_i$ and $t_i[k] \geq t_i^{min}$
5.     Kill the remaining Map tasks ($t_i^{max} - t_i[k]$)
6.     Return the result
```

Fig. 3. *CAB-allocator and CAB-controller* algorithm.

we implement a function `setMaxShare()` in Hadoop's fair scheduler. Using this function, we can adjust the maximum number of containers for each queue.

III. Experimental Results

A. Experimental Setup

We run our experiments on a server with dual-socket Xeon E5-2680 v4 processors, equipped with a total of 28 cores running at 2.4 GHz. The server has 128 GB of DDR4 memory and consumes 350W at maximum load. Ubuntu Server 14.04 with kernel 4.4 is installed on the server with the stock gcc 4.8 and Java 1.7. The server is equipped with Intelligent Platform Management Interface (IPMI) version 2. We use five representative applications from PUMA benchmark suite [2] and some of the Hadoop 2 examples. The five benchmarks are: `Pi-Estimator`, `Top-N`, `Word-Mean`, `Histogram-Movies`, and `Word-Count`. We developed a feedback-based power controller that periodically reads the total AC power using the IPMI interface and adjusts the power cap accordingly using the Intel RAPL controller until the total server power cap is met. The length of sampling window in CAB-estimator is set to five seconds. Our code and benchmarks are available at `github.com/scale-lab/CAB`. We evaluated the performance of *CAB* against three other methods explained as follow:

- *Uncapped Finish Time*: It is the execution time of workloads without any power cap and determines the lower bound of execution time. It processes all the tasks of each job to obtain 100% accurate results. We normalized all the runtime results against the *Uncapped Finish Time*.
- *Capped Finish Time*: It is execution time of workloads under different power caps. It gives the upper bound of execution time of workloads. It also obtains 100% accuracy.
- *Modified ApproxHadoop*: It is a modified version of ApproxHadoop [6]. ApproxHadoop terminates a job after processing the minimum required number of tasks from the job. The modified version behaves the same as ApproxHadoop when the execution time is already greater than the *Uncapped Finish Time*; however, when the *Uncapped Finish Time* has not been reached yet, it continues executing tasks until the *Uncapped Finish Time* is reached. In this way, it can obtain QoR more than the minimum one if there is time and resources.

TABLE I
Specification of jobs in workload sets and the submission queue based on the target QoR accuracy.

		QoR-based Queue Mapping			
	Workload Set	**95% Acc**	**90% Acc**	**85% Acc**	**80% Acc**
	Pi-Estimator	Queue 1	Queue 1	Queue 1	Queue 1
#1	Top-N	Queue 1	Queue 2	Queue 3	Queue 3
	Word-Mean	Queue 3	Queue 3	Queue 3	Queue 3
	Word-Count	Queue 1	Queue 1	Queue 2	Queue 2
#2	Top-N	Queue 1	Queue 1	Queue 3	Queue 3
	Hist-Movies	Queue 3	Queue 3	Queue 3	Queue 3
	Pi-Estimator	Queue 1	Queue 1	Queue 1	Queue 1
#3	Word-Count	Queue 1	Queue 1	Queue 2	Queue 2
	Top-N	Queue 1	Queue 2	Queue 3	Queue 3

B. CAB with power capping

To compare the performance of *CAB* against other methods, we implement several experiments with three job queues. The queues guarantee the execution of 100% of tasks (high-sensitivity queue), 75% of tasks (mid-sensitivity queue), and 50% of tasks (low-sensitivity queue) respectively. We consider three workload sets, each consisting of three benchmark jobs as given in the first column of Table I. In our experiments, we explore four different QoR accuracies: 80%, 85%, 90%, and 95%, where a *QoR target for a workload* ensures that all jobs of the workload meet the target QoR at a minimum. Each job should execute a minimum number of tasks determined by offline profiling, as given earlier in Figure 1, to meet its QoR constraint, and hence, jobs should be submitted to the proper queue that guarantees execution of the minimum number of tasks. As Table I shows, the distribution of the queues changes for different QoR requirements. As an example, the `Top-N` benchmark in the workload set 1 is submitted to Queue 1 for a QoR of 95%, Queue 2 for a QoR of 90%, and Queue 3 for lower QoR targets.

In the first set of experiments, we consider three different power caps: 250W, 200W, and 175W to demonstrate the performance of *CAB* compared to the other methods. In all experiments, power cap is the only changed parameter and all the job and queue specifications are the same. In these experiments, the power cap is fixed and applied throughout the execution of experiments. The normalized runtime of each approach with respect to *Uncapped Finish Time* are shown in Figure 4 for first two workload sets under different power caps and QoR requirements. The results demonstrate that *CAB* mitigates the influence of applied power cap on runtime (*Capped Finish Time*) and reduce it by up to 39.4% (19.18% on average). *CAB* also outperforms the *Modified ApproxHadoop* by up to 10.7% (5.26% on average).

The performance improvements of *CAB* is a function of the importance of resource allocation which depends on the applied power cap. The maximum improvement is reached when the power cap is moderate (e.g., 200 W). In this case, resource reallocation component of *CAB* has a significant role, thus, *CAB* has the maximum improvements compared to the *Modified ApproxHadoop*. *CAB*'s performance improvements decreases when the power cap is too relaxed or too tight. In the case of relaxed power cap (e.g. 250 W), the default allocated resources (equal share) is enough for jobs to either meet the *Uncapped Finish Time* or exceed it slightly. In the case of tight power cap (e.g. 175 W), the available resources are not enough for *CAB* to improve the execution time. Only for Workload set 2 at power 175 W and QoR 95% and 90%,

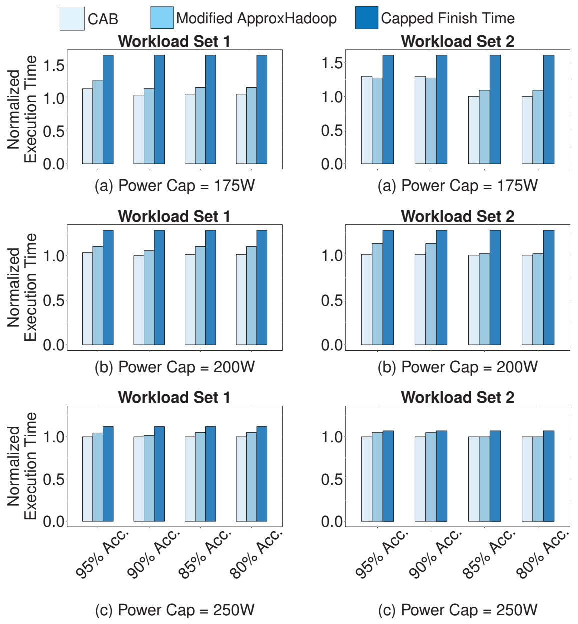

Fig. 4. Normalized execution time of workload set 1 and 2 with respect to *Uncapped Finish Time* approach for achieving desired accuracy under different power caps.

Modified ApproxHadoop performs insignificantly better than *CAB* because of small errors in finish time estimation.

In the second experiment, we illustrate the operation of *CAB* as the applied power caps change dynamically during the time. We use Workload set 3 in this experiment. Using a target QoR of 85% for illustration, Figure 5 provides the details of how *CAB* manages the resources (i.e., core-based containers) among the queues as the power cap changes dynamically. At the beginning, all three queues start with equal amount of resources and *CAB* starts estimating the finish time of the jobs. After the appropriate sampling window, *CAB* reallocates the resource shares among the job queues based on current progress rate of jobs, their minimum required tasks, and current power cap. Since the job submitted to Queue 1 (`Pi-Estimator`) should finish all its tasks successfully, *CAB* gives the most amount of resources to this queue, and when the power cap becomes tight, it gives even more resources to it to compensate for the impact of the power cap on the progress rate of the job. *CAB* allocates less resources to Queue 2 since this queue is less sensitive, but still more than Queue 3. Finally, since Queue 3 is the least sensitive one and its progress rate is more than what it needs to reach minimum QoR without runtime stretch, *CAB* gives the least amount of resources to this queue. Upon completion of a job, *CAB* distributes the resources of its queue among the other queues, so they can finish earlier. The time gap between applying a power cap and resource allocation adjustment is due to 1) time needed for the RAPL controller to adapt to the new power cap, and 2) the time needed for *CAB* to detect the change of progress rate caused by the new power cap. Since the *CAB* checks the progress rates periodically and adjusts the resource shares according, *CAB* might change the resource share of queues when caps do not change. We repeat the experiment with different QoR targets and summarize the results in Table II. We also contrast the performance of *CAB* against *Modified ApproxHadoop*, where it is able to outperform it by up to 8.03%, and reduce the impact of power capping on runtime

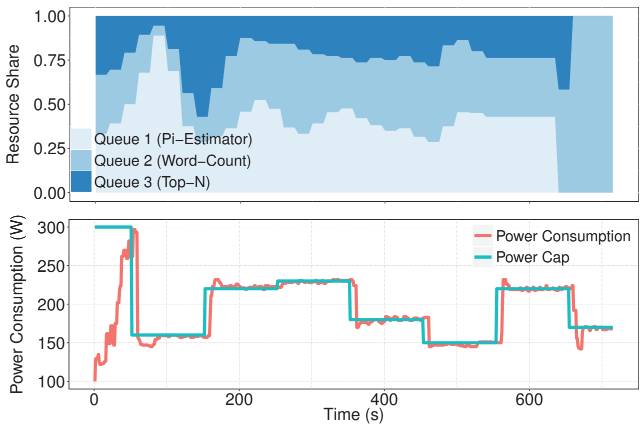

Fig. 5. *CAB* resource management of different queues according to minimum required tasks of each queue under dynamic power cap.

TABLE II
SUMMARY OF RESULTS FOR WORKLOAD SET 3 UNDER DYNAMIC POWER CAPS (NORMALIZED TO UNCAPPED FINISH TIME).

Approach	Desired accuracy			
	95% Acc	90% Acc	85% Acc	80% Acc
CAB	1.72	1.42	1.22	1.22
Modified ApproxHadoop	1.72	1.55	1.29	1.29
Capped Finish Time	1.72	1.70	1.65	1.65

(*Capped Finish Time*) by 26.1%. For 95% accuracy, we see that all the approaches have the same performance. The reason is that when the QoR requirement is high, then there is not much benefit from using approximation techniques since most tasks (all the tasks in this case) have to be executed anyway.

ACKNOWLEDGMENT

We thank Dr. Ricardo Bianchini from Microsoft Research for his helpful feedback. This work is partially supported by NSF grants #1420864 and #1438958, and Sharif University of Technology grant G930826.

REFERENCES

[1] Datacom equipment power trends and cooling applications. *ASHRAE*, 2005.
[2] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar. Puma: Purdue mapreduce benchmarks suite. 2012.
[3] R. Azimi, M. Badiei, X. Zhan, N. Li, and S. Reda. Fast decentralized power capping for server clusters. In *HPCA*, pages 181–192, 2017.
[4] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar. The need for speed and stability in data center power capping. *Sustainable Computing: Informatics and Systems*, 3(3):183–193, 2013.
[5] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & cap: adaptive dvfs and thread packing under power caps. In *IEEE/ACM MICRO'11*, pages 175–185, 2011.
[6] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen. Approxhadoop: Bringing approximations to mapreduce frameworks. In *SIGARCH Comp. Architecture News*, volume 43, pages 383–397, 2015.
[7] D. R. Krishnan, D. L. Quoc, P. Bhatotia, C. Fetzer, and R. Rodrigues. Incapprox: A data analytics system for incremental approximate computing. In *WWW'16*, pages 1133–1144, 2016.
[8] S. Mitra, M. K. Gupta, S. Misailovic, and S. Bagchi. Phase-aware optimization in approximate computing. In *IEEE/ACM CGO'17*, pages 185–196, 2017.
[9] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *ISCA'06*, pages 66–77, 2006.
[10] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song. Dynamo: facebook's data center-wide power management system. In *ISCA'16*, pages 469–480, 2016.
[11] X. Zhan and S. Reda. Power budgeting techniques for data centers. *IEEE Trans. on Computers*, 64(8):2267–2278, 2015.