TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 03/10 pp24-32
Volume 19, Number 1, February 2014

Towards a Cost-Efficient MapReduce: Mitigating Power Peaks for Hadoop Clusters

Nan Zhu, Xue Liu*, Jie Liu, and Yu Hua

Abstract: Distributed data processing system is becoming one of the most important components for data-intensive computational tasks in the enterprise software infrastructure. Deploying and operating such systems require large amount of costs, including hardware costs to build clusters and energy costs to run clusters. To make these systems sustainable and scalable, power management has been an important research problem. In this paper, we take Hadoop as an example to illustrate the power peak problem which causes power inefficiency and provides in-depth analysis to explain issues with existing system designs. We propose a novel power capping module in the Hadoop scheduler to mitigate power peaks. Extensive simulation studies show that our proposed solution can effectively smooth the power consumption curve and mitigate temporary power peaks for Hadoop clusters.

Key words: power peaks; power management; MapReduce

1 Introduction

Distributed data processing system is becoming one of the most important components in enterprise software infrastructure. In order to improve system performance and scalability, MapReduce[1], as well as its open-source implementation, Hadoop, has been widely used to support large-scale computation tasks over big datasets. For example, America Online leverages MapReduce to analyze users' behaviors[2]; Facebook builds the system infrastructure on Hadoop to handle large volume of requests from Internet[3]; By running large-scale machine learning tasks on MapReduce, WalMart exploits the context of users' online posts to push proper product information[4]. These MapReduce systems consume substantial system resources, e.g., power, on large-scale physical or virtualized clusters to fully exploit the parallelism of computation tasks on MapReduce.

In order to deliver high performance, large-scale data centers have to deal with power-related problems due to large power consumption and electricity bills. For example, Google's deep learning neural net[5] consisting of 16 000 CPUs incurs a $1 million cost in hardware. Amazon's researcher pointed out that companies have to pay much more on operational costs comparing with hardware costs in servers. This operational cost is dominated by *Power Cost*[6]. High power cost results in improving the scalability of MapReduce systems ever more difficultly. Moreover, in order to efficiently handle the peak load, we need to subscribe Power Distribution Units (PDUs), as well as related power provisioning infrastructure in higher-level configurations. This solution can support significant performance improvements when the peak loads occur.

Power capping technique can mitigate power peaks caused by peak loads and alleviate overload cases in data centers[7]. Power capping can obtain the design goals in terms of power efficiency and system performance improvements. Specifically, in order to

• Nan Zhu and Xue Liu are with the School of Computer Science, McGill University, Montreal, H3A 0G4, Canada. E-mail: nan.zhu@mail.mcgill.ca; xueliu@cs.mcgill.ca.
• Jie Liu is with Microsoft Research, Redmond, WA 98052-6399, USA. E-mail: Jie.Liu@microsoft.com.
• Yu Hua is with WNLO, School of Computer, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: csyhua@hust.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2013-12-02; revised: 2014-01-01; accepted: 2014-01-02

guarantee the limited consumption of energy, we can judiciously schedule the computation tasks in the servers. The power capping can reduce the execution costs and improve the scalability of MapReduce systems. The conventional power capping scheme unfortunately does not work well for MapReduce workloads based on our observations in Microsoft Production Cluster[8,9].

In this paper, we comprehensively analyze the task scheduling mechanism in Hadoop and gain insights of the correlation between the server load and the power consumption in each server. Based on our observations, we implement a power capping module in the Hadoop scheduler based on feedback control theory to support power-aware task scheduling. In our implementation, we construct a dynamic model of the power consumption on each server and an evaluation model for each control decision based on the utilization of power budget. The evaluation result from the control decisions is used to adaptively adjust the parameters in the server power model and support more accurate decision in the next control phase. In this way, our model can not only capture the dynamics of workloads in the MapReduce system[10], but also obtain a suitable tradeoff between the system performance and power efficiency. Moreover, the improved systems based on Hadoop, like Spark[11], have the similar scheduling method. We hence believe that our observation is also applied to other distributed data processing systems. System developers will benefit from our proposed schemes.

In this paper, we make the following contributions:

- We make a detailed analysis on Hadoop scheduling mechanism. From the comprehensive analysis, we observe the correlation between the power peak and scheduling schemes.
- We leverage a power capping module into Hadoop scheduler, which offers much more efficient control on power consumption than traditional capping techniques.
- We build a novel simulator, i.e., PowerMumak, to simulate the execution of the MapReduce cluster and evaluate the performance of our adaptive approach.

2 Analysis on Hadoop Scheduling and Power Consumption

In this section, we comprehensively study the Hadoop scheduling strategy, as well as identifying the problem of power peaks.

2.1 Computation in Hadoop

In order to deliver high performance, Hadoop needs to efficiently manage and schedule the computation jobs. Most of Hadoop jobs are divided into two phases, *Map* and *Reduce* (some jobs only contain *Map* phase). Each phase consists of one or more concurrent task(s). When both phases exist, the output of Map is the input of Reduce. Users describe their computation logics on Map and Reduce tasks by calling Hadoop APIs with high-level programming languages, like Java, Python[12] or more expressive language like HiveQL (SQL-Like)[13]. The user-specific computation logic locates in the Map() and Reduce() functions of Map and Reduce phase respectively.

Each Map task is assigned via a split of input file, called *Data Block* or *Data Chunk*. Its size is usually 64 MB or 128 MB, which can be defined by a distributed file system. The Map task first reads the split file into the memory, and uses Map() function to Data Block, yielding the results in the form of key-value pairs. When this operation is completed, the Map task notifies the Hadoop *JobTracker*, which then directs the Reduce tasks to fetch the Map tasks' outputs and process them.

Each Reduce task is divided into 3 stages, *Shuffle*, *Sort*, and *Reduce*.

- **Shuffle**: Each reduce task is associated with a partition of key range generated by the Map tasks. It uses the assigned keys to selectively pull the input data from the servers that maintain the outputs of the Map task.
- **Sorting**: In this stage, tasks can be grouped into the key-value pairs based on the keys.
- **Reduce**: The execution of user-specific computation logic, Reduce(), can produce the final output of the job.

The analysis in this subsection shows two observations: (1) The fine-grained usage pattern of server resources, like CPU, memory, and NIC, is defined by Map() and Reduce() functions. They are unpredictable and uncontrollable since they are completely decided by users' requirement; (2) Based on (1), we can only have a coarser-grained control on the usage of server resources to mitigate the power peaks. We will leverage the analysis on the default job/task scheduling in Hadoop to show the rationale of our solution to achieve power efficiency.

2.2 Scheduling of MapReduce tasks

The scheduling framework has been improved in the development of Hadoop. We first discuss the design used in Hadoop 1.x and earlier versions, and then reveal that our analysis is still valid under the new scheduling design (Hadoop 2.x version).

There are two kinds of processes managing the jobs and tasks in Hadoop 1.x as follows.

- **JobTracker:** JobTracker is a process to accept job requests from clients. It registers the job requests with the system, e.g., creating job objects, assigning JobID, allocating system resources to them, etc. It divides the new accepted job into tasks and pushes the tasks into the pending queue of the system. The JobTracker communicates with *TaskTracker* via periodic *HeartBeat* message, through which it tracks the existence of TaskTrackers and monitors the progress of running tasks. Moreover, the tasks are dispatched to TaskTrackers by JobTracker. The scheduling algorithm of tasks is implemented as *TaskScheduler* that is a module of JobTracker.
- **TaskTracker:** In a typical Hadoop deployment, there are always multiple *TaskTrackers* that accept task assignments from JobTracker and execute the tasks. Each TaskTracker has a fixed number of *MapSlots* and *ReduceSlots* to accept Map and Reduce tasks respectively. Each slot can contain no more than 1 task. TaskTracker reports the number of tasks it can afford to JobTracker through *Heartbeat message*. For Map task, the JobTracker assigns as many as possible upon receiving the report from a TaskTracker. For Reduce tasks, the TaskTracker can obtain at most one reduce task in a single HeartBeat even there are more than one available ReduceSlots. The conservative scheme is to avoid the network congestion on the server-end caused by multiple Reduce tasks fetching the Map output simultaneously.

From the above analysis, we can observe that the total server resource utilization in a cluster is determined by how many concurrent tasks are scheduled as running by JobTracker. In the meantime, the utilization distribution of all servers is determined by how JobTracker dispatches tasks to TaskTrackers. To implement a cost-efficient MapReduce/Hadoop system, the JobTracker is designed to be power-aware to support accurate scheduling decision.

In Hadoop 2.x[12], the JobTracker is split into multiple *ApplicationMaster*s (one per Hadoop job) and a *ResourceManager*. The ApplicationMasters negotiate with ResourceManager to obtain resources—in terms of a number of *Containers*—to execute tasks. The changes on the architecture do not affect the validation of our analysis. We investigate the ResourceManager implementation in Hadoop 2.x, and identify that ResourceManager can still meet the needs of resources from ApplicationMasters. In this paper, our analysis and implementations are based on Hadoop 1.x version, which can be easy to extend to 2.x version by porting the algorithm from JobTracker to ResourceManager.

2.3 Power peak problem in Hadoop clusters

We observe the data from a 200-node simulated Hadoop[14] cluster. The power consumption of the cluster is shown in Fig. 1 (More details are in Section 2.4). The power consumption curve starts at around 10 kW, which is close to the idle power of the whole cluster. It spends about 210 virtual minutes for the cluster to complete all workloads. During the simulation time, the curve surges at a few time intervals. According to the power consumption distribution, there are less than 8% time intervals where the cluster is close to produce its peak power. These peaks are not mitigated by the used capping facilities in the cluster. In this paper, we argue that the capping module can be embedded into the scheduling module, instead of only relying on the existing techniques like DVS[15]. Our evaluation result shows that this embedded module brings much more efficient control on the power consumption.

2.4 Analysis on the power consumption using JobQueueTaskScheduler

JobQueueTaskScheduler is the default task scheduler implemented in Hadoop 1.x version. It schedules tasks following First-Come-First-Serve (FCFS) principle, i.e., the later arrived jobs can only be served after

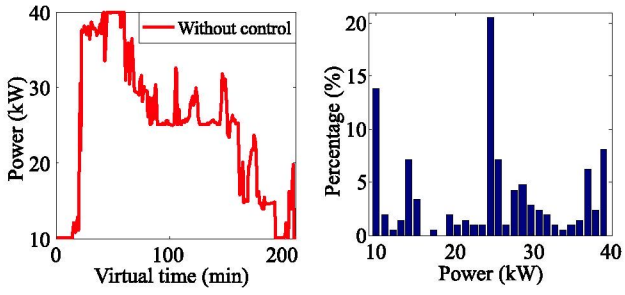


Fig. 1 Power consumption and its distribution in the Hadoop cluster.

all tasks of the earlier jobs are scheduled. In this subsection, we illustrate the power peak problem in Microsoft Production MapReduce cluster by simulations. Our goal is to reveal how the default scheduling setup in Hadoop results in power peaks, which incurs expensive costs on the scalability of the cluster.

We elaborate the power peak problem in a 200-node Hadoop cluster, in which every computing server has 8 processing cores with the frequency of 2.4 GHz. We use the workload trace specified in Table 1. Jobs are distinguished by Map and Reduce task numbers, and we also indicate the number of each kind of job.

We first show that the scheduling of tasks is correlating with the power peaks of the Hadoop cluster. We track both the power consumption and the number of concurrent tasks in the system with the sampling frequency of 1 min. We present the power consumption and tasks scheduling trace in Fig. 2. We can observe that the power consumption surges with the increase of the task number in most cases while there are some "outliers" (more analysis about these outliers can be found in our early work[9]).

Table 1 Trace file.

Number of Map tasks	Number of Reduce tasks	Number of Jobs
1	500	6
3	1	1
20	1	3
35	101	1
101	1	1
131	47	1
329	101	1
500	1	3
3204	0	2

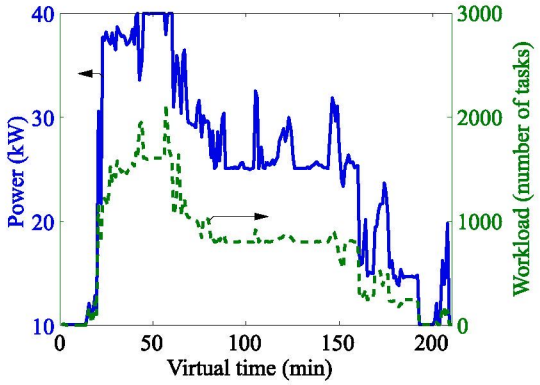


Fig. 2 Power consumption vs. workload scheduling based on virtual time.

From Fig. 2, we observe that the default scheduling method in the Hadoop system leads to poor system performance. At a few time intervals the cluster consumes the high power, and during most of the total time intervals the cluster operates at the low power. As described before, these rarely-achieved high peaks introduce high costs on scaling Hadoop cluster. With the analysis in the previous subsections, we argue that the default scheduling policy of Hadoop is the main reason to the unexpected peaks: The scheduler in JobTracker assigns a new task to TaskTracker whenever it has a free task slot, despite the fact that the server on which TaskTracker run could have already been very close to overload condition. When all or the most of servers in the cluster are on the overloaded status, the peaks in total power consumption appear. Other schedulers[16,17] working in the multi-tenant Hadoop environment cannot address this problem, since the task is still scheduled upon a TaskSlot. Due to the space limitation, we do not show our observations on those schedulers here since they are similar with the results shown in Fig. 2.

It is desirable that the system is utilized by eliminating the high power consumption. To achieve this goal, we propose an adaptive power-aware scheduling approach.

3 Power-Aware Hadoop Scheduler

In this section, we present the design of our power-aware Hadoop scheduler. We start with discussing the design considerations and then we demonstrate the architecture of our scheduler and discuss how our schedule handle dynamic workload and trade-off between power efficiency and performance respectively.

3.1 Design considerations

There are some other technologies besides power capping for reducing the power consumption in MapReduce cluster, e.g., sleeping the idle servers[18]. The sleeping operation requires some costs. Restarting the servers consumes time and for most of MapReduce systems, the distributed file system and MapReduce share the same group of physical nodes. Shutting down the servers may reduce the number of available copies of data chunks and cause the re-balance operation in the distributed file system, thus leading to the performance degradation. Meisner et al. obtained the similar conclusion in Ref. [19].

Other ways, like BEEMR[20], leverage the

sophisticated data placement scheme of distributed file systems. These scheme may not be available to other cases. For example, in BEEMR, it assumes that MIA jobs read data only from a limited number of machines in the cluster, but in our environment and Google's datacenter[19], the MapReduce jobs can use all machines in the data center.

Existing power capping technique does not meet the needs of MapReduce system. The power models are static for each server, which assumes that the characteristics of the workloads used in the servers exhibit few changes. For MapReduce, the usage pattern of the clusters does change. Table 1 shows that the number of Map tasks in a mixed workload ranges from 1 to 3204, and the Reduce tasks ranges from 0 to 500. Hence we need to dynamically update the model of the whole system and leverage an adaptive way for power management.

The power capping achieves power efficiency based on the premise that the performance does not suffer too much for power management. We hence need to design a systematic approach in the scheduler to capture the trade-off.

3.2 Design of power-aware Hadoop scheduler

In this section, we present the design of our power-aware scheduler. The scheduler inherits most of functionalities of JobQueueTaskScheduler but executes an additional admission control logic when receiving the Heartbeat from TaskTrackers.

The workflow of the admission controller is shown in Fig. 3. The controller mitigates the power consumption in the cluster by adjusting the available amount of new running tasks in each control period. The controller consists of two modules: the model estimator and the controlling module. The model estimator dynamically models the power consumption of each server to ensure the accuracy under the dynamic workloads. In order to manage power peaks, the controller module makes control decisions based on the model generated by the model estimator. Power Tracking module is used to measure the power consumption of the Hadoop cluster, and its measure results will be sent back to the controller.

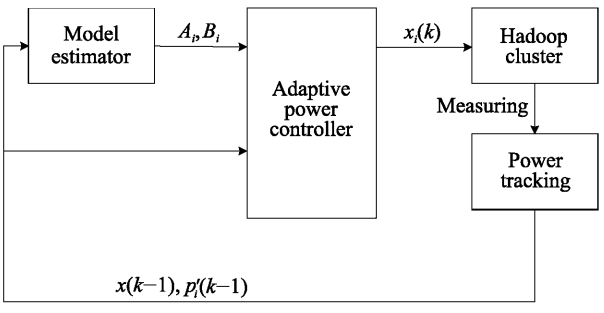


Fig. 3 The design of power controller.

We model the power consumption of each server i as follows:

$$p_i(k) = A_i p_i'(k-1) + B_i x_i(k) \tag{1}$$

where A_i and B_i are the unknown system parameters and these parameters may vary due to the dynamic workloads; p is the estimated value of power consumption; p' is the measured power consumption; x_i represents the allowed number of concurrent tasks for node i; k is the timestamp for each control period. In this design, we only capture the power consumption in the last control period, and the input trajectory consists of a single value. This decision simplifies our implementation and can perform well in practice.

3.3 Control power peaking

To achieve the control goal with power capping, we define the following cost function for each server i. We transfer the goal of limiting power drawing to minimizing the value of cost function:

$$J_i(k) = \left(p_i(k+1) - p_i^{\text{cap}}(k+1)\right)^2 \tag{2}$$

Minimizing the cost function indicates that the gap between the consumed power and the power cap value should be as small as possible. This cost function can capture the trade-off between the performance and the power consumption at the same time. Minimizing the function means that we do not want the power consumption over the threshold or at least keep the gap falling into an acceptable range. Furthermore, the minimized gap indicates that we need to fully utilize the given power budget.

3.4 Capture dynamic workload

As the dynamic workload produces different usage patterns in the server resources, the relationship between x_i and p_i may change. To ensure the accuracy of the model, the model estimator obtains the generated value x_i from the controller and the actual consumed power value from the power tracking module at each control period, and computes the new system parameters A_i and B_i. We use the Recursive Least Square (RLS) estimator with exponential forgetting to identify the system parameters A_i and B_i for all servers. The model estimator sends the updated system

parameters to the controller, which will be used in the next control decision.

We rewrite the system model in the following RLS-friendly form for each node i, which we use in the remainder of the paper.

$$p_i(k) = \boldsymbol{\theta}_i(k)\boldsymbol{\varphi}_i(k) \tag{3}$$

where

$$\boldsymbol{\theta}_i(k) = [A_i, B_i] \tag{4}$$

$$\boldsymbol{\varphi}_i(k) = [p_i(k-1), x_i(k-1)]^{\mathrm{T}} \tag{5}$$

RLS estimator can be used to identify the time varying parameter matrix $\boldsymbol{\theta}_i$ online. This estimator has been applied extensively in adaptive control system design as it can converge and reject disturbance in an efficient manner. The estimator is described by the following equations:

$$\begin{aligned} \varepsilon_i(k) &= P_i(k) - \hat{\boldsymbol{\theta}}_i(k-1)\boldsymbol{\varphi}_i(k), \\ \hat{\boldsymbol{\theta}}_i(k) &= \hat{\boldsymbol{\theta}}_i(k-1) + \frac{\varepsilon_i(k)\boldsymbol{\varphi}_i^{\mathrm{T}}(k)X_i(k-1)}{\lambda + \boldsymbol{\varphi}_i^{\mathrm{T}}(k)X_i(k-1)\boldsymbol{\varphi}_i(k)}, \\ X_i(k) &= \frac{1}{\lambda}\Big[X_i(k-1) - \\ &\frac{X_i(k-1)\boldsymbol{\varphi}_i(k)\boldsymbol{\varphi}_i^{\mathrm{T}}(k)X_i(k-1)}{\lambda + \boldsymbol{\varphi}_i^{\mathrm{T}}(k)X_i(k-1)\boldsymbol{\varphi}_i(k)}\Big] \end{aligned} \tag{6}$$

where $\hat{\boldsymbol{\theta}}_i(k)$ is the estimation of the true value of model parameter $\boldsymbol{\theta}_i(k)$ at the k-th control point, λ is the forgetting factor. In practice, λ typically has a positive value between 0.97 and 0.995.

4 Performance Evaluation

We evaluate our proposed approach with a new Hadoop simulator. Extensive evaluation results demonstrate that our proposed method can efficiently address the problem of power peak.

4.1 Hadoop simulator

To support our study on addressing the problem of temporary power peaks and evaluating the proposed solution, we develop a new simulator called PowerMumak to replay the workload trace file. The design of PowerMumak augments Mumak[21] and implements the functionalities of power module in JobTracker and TaskTracker. The architecture of PowerMumak is shown in Fig. 4. Rumen[22] is a tool to generate JSON formatted Hadoop cluster trace from the job running log files.

This simulator can exactly simulate the working mechanism of Hadoop scheduling framework. The simulated scheduler class extends the JobTracker implementation in Hadoop by adding interaction module with the discrete event simulation engine. It uses the same strategy to implement the simulated TaskTracker. In this way, when a simulated task is to be scheduled, the simulator calls the same function in the real JobTracker implementation. Our simulator is effective to reproduce the behaviors of the real Hadoop scheduler.

Without the loss of generality, we illustrate our proposed method with a power model for computation-intensive workloads[23] and use a power function in terms of the processor's frequency and utilization. The power model can be extended to exploit more types of workloads into considerations and we can extend it in our design. Our model is described as

$$p_i' = a_{i3}f_i u_i + a_{i2}f_i + a_{i1}u_i + a_{i0} \tag{7}$$

where the system parameters a_{ij} $(j = 0, 1, 2, 3)$ can be determined by system identification of physical servers; p_i, f_i, and u_i respectively represent the power consumption, processor's frequency, and utilization of

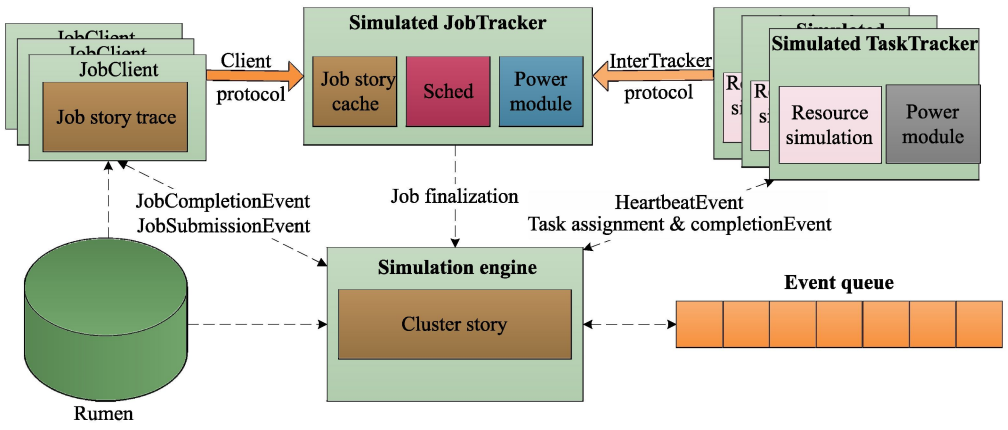


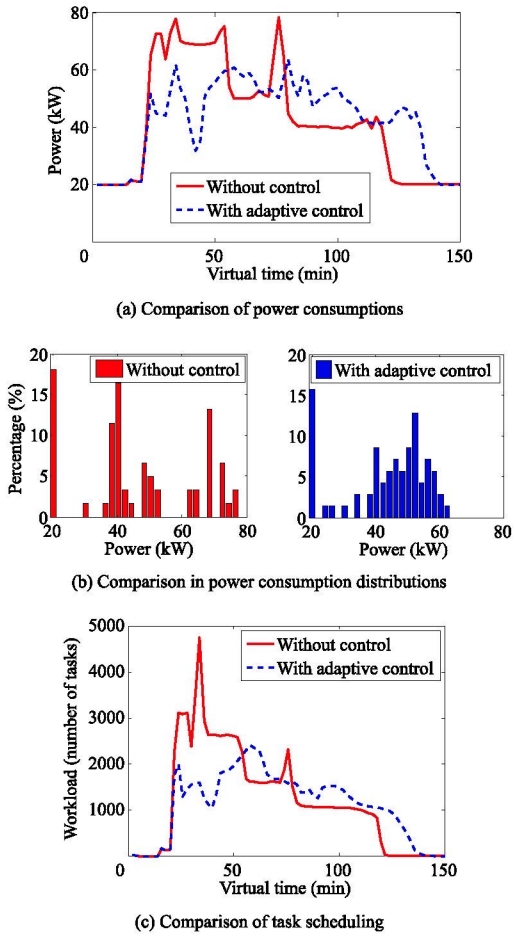
Fig. 4 The architecture of PowerMumak.

node i. We use the steady-state result of the M/M/n queueing model to model each core in the server. The node utilization can be described as $u = \frac{x}{sn}$, where x is the number of concurrent tasks in the current sampling cycle, s is the number of the served tasks, and n is the server's core number. In this paper, we assume that all servers are homogeneous with parameters: $a_{i3} = 68.4$, $a_{i2} = 14.6$, $a_{i1} = -14.2$, $a_{i0} = 15.0$.

4.2 Power consumption and workload scheduling

We replayed the workload trace specified in Table 1. This workload trace contains the typical workload in a data center, where the jobs containing small number of Map tasks and Reduce tasks usually handle the interactive query requests, and the jobs with high parallelism are usually seen when the backup data service is running. We illustrate the simulation results with a 400-node Hadoop cluster, in which each node has 8 processing cores and the frequency of 2.4 GHz. We set the power capping reference for the entire cluster as 64 kW and assume that each server in the Hadoop cluster has the same power reference of 160 W. In the real implementations, the power capping reference for the cluster can be determined by experienced domain experts.

The results are shown in Fig. 5. According to Fig. 5a, the power peaks have been mitigated and the power consumption curve is smoothed with the proposed adaptive control method. To better analyze Fig. 5a, we calculate the power consumption distributions as shown in Fig. 5b. We can observe that the power distributions are high around 20 kW. By the comparison with power consumption distributions, we observe that the power peaks have been effectively mitigated. At the same time, the cluster mainly works at the power level between 40 and 60 kW. By comparing the workload scheduling traces in Fig. 5c, we can observe that the workload execution time (140 virtual minutes) under the adaptive control is slightly longer than that (128 virtual minutes) without control. Under a mixed workload, our solution can reduce the power budget for building data center with 20%, i.e., 16 kW, bringing less than 10% latency on execution. Under the given power budget, we can use another 100 machines (25% of the current size) in the data center.



(a) Comparison of power consumptions

(b) Comparison in power consumption distributions

(c) Comparison of task scheduling

Fig. 5 Power consumptions, distributions and workload scheduling based on virtual time.

5 Related Work

Energy efficiency in the cloud computing has attracted much attention. Fan et al.[7] analyzed the aggregate power usage characteristics of large scale computing cluster under different workload patterns and presented the possibility of reducing peak consumption by power capping technique. Dynamic voltage scaling[15] is used in multi-tier server architecture. The analysis on the Online Data-Intensive (OLDI) services workload[19] can help evaluate the possible solutions to the power management. The work in Ref. [24] characterizes the energy efficiency of various workloads in Hadoop. The sleepy strategy[18] is used to save power for Hadoop cluster. Unlike them, we analyzed that restarting the server introduces time cost and the performance in the underlying file system service. In addition, the MapReduce jobs are only related with some parts of the cluster[20], which is not the case in Microsoft Production Cluster.

6 Conclusions

Motivated by the power peak problem observed from Microsoft Production Servers, we investigate this

problem in this paper. We propose the design of an adaptive scheme to efficiently manage the power peaks for MapReduce clusters. By using the simulation, we evaluate the performance of addressing the power peak problem. Extensive simulation results show that our proposed methods can effectively smooth the power consumption curve and mitigate temporary power peaks for MapReduce clusters. We hence can offer scalability for our MapReduce cluster with small costs. Our design can be used in distributed data processing systems.

Acknowledgements

Xue Liu would like to thank the support of the National Science Foundation of USA (No. 1116606). Yu Hua would like to thank the support of the National Nature Science Foundation of China (No. 61173043).

References

[1] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, in *Operating Systems Design and Implementation (OSDI) 04'*, 2004, pp. 137-150.

[2] Hadoop, Hadoop poweredby, http://wiki.apache.org/hadoop/PoweredBy, June 2013.

[3] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer, Apache Hadoop goes realtime at facebook, in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2011, pp. 1071-1080.

[4] M. van Rijmenam, Walmart makes big data part of its dna, http://smartdatacollective.com/bigdatastartups/111681/walmart-makes-big-data-part-its-social-media, March 2013.

[5] G. Anthes, Deep learning comes of age, *Commun. ACM*, vol. 56, no. 6, pp. 13-15, 2013.

[6] J. Hamilton, Cost of power in large-scale data centers, http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx, November 2008.

[7] X. Fan, W. D. Weber, and L. A. Barroso, Power provisioning for a warehouse-sized computer, in *Proceedings of the ACM IEEE International Symposium on Computer Architecture (ISCA)*, 2007.

[8] N. Zhu, L. Rao, X. Liu, and J. Liu, Handling more data with less cost: Taming power peaks in MapReduce clusters, in *Proceedings of the Asia-Pacific Workshop on Systems, APSYS '12*, New York, NY, USA, 2012.

[9] N. Zhu, L. Rao, X. Liu, J. Liu, and H. Guan, Taming power peaks in MapReduce clusters, in *Proceedings of the ACM SIGCOMM 2011 Conference, SIGCOMM'11*, New York, NY, USA, 2011, pp. 416-417.

[10] Y. Chen, S. Alspaugh, and R. H. Katz, Design insights for MapReduce from diverse production workloads, Technical Report UCB/EECS-2012-17, EECS Department, University of California, Berkeley, USA, 2012.

[11] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, Spark: Cluster computing with working sets, in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10*, Berkeley, CA, USA, 2010, p. 10.

[12] T. White, *Hadoop: The Definitive Guide*, 3rd Edition. O'Reilly Media, Inc., 2012.

[13] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, Data warehousing and analytics infrastructure at facebook, in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD'10*, New York, NY, USA, 2010, pp. 1013-1020.

[14] Hadoop, http://hadoop.apache.org/, 2013.

[15] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, Dynamic voltage scaling in multitier web servers with endto-end delay control, *IEEE Trans. Comput.*, vol. 56, no. 4, pp. 444-458, 2007.

[16] Hadoop, Capacityscheduler, http://hadoop.apache.org/docs/stable/capacity/scheduler.html, June 2013.

[17] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, Job scheduling for multi-user mapreduce clusters, Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley, USA, April 2009.

[18] J. Leverich and C. Kozyrakis, On the energy (in)efficiency of hadoop clusters, presented at Workshop on Power-Aware Computing and Systems (HotPower), 2009.

[19] D. Meisner, C. M. Sadler, L. A. Barroso, W. D.Weber, and T. F.Wenisch, Power management of online data-intensive services, in *Proceedings of the ACM IEEE International Symposium on Computer Architecture (ISCA)*, 2011.

[20] Y. Chen, S. Alspaugh, D. Borthakur, and R. H. Katz, Energy efficiency for large-scale mapreduce workloads with significant interactive analysis, in *Proceedings of the 7th ACM European Conference on Computer Systems*, New York, NY, USA, 2012.

[21] Apache software foundation, Mumak: MapReduce simulator, https://issues.apache.org/jira/browse/MAPREDUCE-728, 2013.

[22] Hadoop, Rumen, http://hadoop.apache.org/docs/stable/rumen.html, August 2013.

[23] T. Horvath and K. Skadron, Multi-mode energy management for multi-tier server clusters, presented at the Seventeenth International Conference on Parallel Architectures and Compilation Techniques (PACT), 2008.

[24] Y. Chen, L. Keys, and R. H. Katz, Towards energy efficient MapReduce, Berkeley Technical Report EECS-2009-109, 2009.

Nan Zhu is a PhD candidate in School of Computer Science, McGill University, Canada. He received his BS and MS degrees from Nanjing Institute of Technology and Shanghai Jiaotong University in 2005 and 2012, respectively. His research interests include distributed system, large-scale data processing system, and computer networks (especially software-defined networking). His work has been presented in SIGCOMM 2011, ApSys 2012, ICAC 2013, etc. He currently focuses on improving resource management strategy in interactive data analytic system.

Xue Liu is an associate professor in the School of Computer Science at McGill University, Canada. He received his PhD degree in computer science from the University of Illinois at Urbana-Champaign in 2006. He has also worked as the Samuel R. Thompson Associate Professor in the University of Nebraska-Lincoln and HP Labs in Palo Alto, California. His research interests are in computer and communication networks, real-time and embedded systems, distributed systems, cyber-physical systems, green computing, and smart grid. He has published more than 150 research papers in major peer-reviewed international journals and conference proceedings in these areas. His research received the Year 2008 Best Paper Award from the IEEE Transactions on Industrial Informatics, and the First Place Best Paper Award from the ACM Conference on Wireless Network Security 2011 (WiSec 2011). Dr. Liu's research has been reported by news media including the New York Times, Computer World, The Register, Huffington Post, CBC, NewScientist, MIT Technology Review's Blog, and McGill Daily, etc. He has been granted 2 US patents and filed 2 other US patents. He is a recipient of the Tomlinson Scientist Award from McGill University. He serves on the editorial board of IEEE Transactions of Parallel and Distributed Systems, IEEE Transactions on Vehicular Technology, and IEEE Communications Surveys and Tutorials.

Jie Liu is a Principal Researcher at Microsoft Research, Redmond, WA, and the manager of its Sensing and Energy Research Group. He received his PhD degree in electrical engineering and computer sciences from UC Berkeley in 2001, and his MEng and BEng degrees from Tsinghua University, China. From 2001 to 2004, he was a research scientist in Palo Alto Research Center (formerly Xerox PARC). From 2008, he has been an adjunct professor at Harbin Institute of Technology, China. His research interests root in understanding and managing the physical properties of computing. He has published broadly in areas like sensor networks, embedded systems, ubiquitous computing, and data center energy management. He also holds more than 40 patents in these fields. He is an associate editor of ACM Trans. on Sensor Networks, was an associate editor of IEEE Trans. on Mobile Computing, and has chaired a number of top-tier conferences. Among other recognitions, he received Best Paper Awards in SenSys 2011 and RTAS 2010, the Leon Chua Award from UC Berkeley in 2001, Technology Advance Award from (Xerox) PARC in 2003, and a Gold Star Award from Microsoft in 2008. He is an ACM Distinguished Scientist.

Yu Hua received his BEng and PhD degrees in computer science from Wuhan University, China, in 2001 and 2005, respectively. He is an associate professor at the Huazhong University of Science and Technology, China. His research interests include computer architecture, cloud computing, and network storage. He has more than 50 papers to his credit in major journals and international conferences including IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, USENIX ATC, INFOCOM, SC, ICDCS, ICPP, and MASCOTS. He has been on the program committees of multiple international conferences, including INFOCOM and ICPP. He is a senior member of the IEEE, and a member of ACM and USENIX.