Green MapReduce for Heterogeneous Data Centers

Derya Çavdar*†, Lydia Y. Chen*, Fatih Alagoz†
IBM Research Zurich Lab, Switzerland*
{avd, yic}@zurich.ibm.com
Bogazici University†
{fatih.alagoz}@boun.edu.tr

***Abstract*—MapReduce has emerged as one of the key workloads in today's data centers, which constantly strive for an optimal tradeoff between energy consumption and performance. MapReduce alternates between computation and communication intensive phases with bursty workloads. The challenge to make execution of MapReduce green, lies in controlling server and network resources simultaneously. The related work offers various good solutions for homogenous systems, with the central theme of packing tasks into as small number of servers as possible and thus overlooking the possibility to "sleep" servers and network components. This paper considers a very bursty MapReduce workload with distinct CPU, memory and network requirements executed on heterogenous data centers, where servers have various CPU/mory capacities and execute request in a process-sharing manner. To reduce energy consumption while maintaining a low task response time, we propose an online energy minimization path algorithm, termed GEMS, to schedule MapReduce tasks, in cooperation with sleeping policies on servers as well as the switches. Using Google MapReduce traces, our simulation experiments show that our proposed solution gains a significant energy saving of 35% and meanwhile improves task response times by 35% on heterogenous data centers, compared to policies which are network agnostic or adopt no sleeping schedule. Overall, we achieve greener and faster MapReduce with (surprisingly) only a slightly higher number of servers, by considering energy consumption rather than conventional approach of considering power values only.**

***Index Terms*—MapReduce, energy optimization, heterogeneous data centers**

I. Introduction

MapReduce application is an increasingly important class of datacenter workloads, featuring on a powerful and efficient computation of complex queries that involve a huge amount of data. These applications alternate between map and reduce phases, where tasks are processed on distributed computing units and periodically communicate results to the master tasks in an all-to-one fashion. The response time of tasks depends not only on the processing capacities of data centers, but also the network communication that is shown particularly influential for the tail response times [1]. Moreover, arrivals of MapReduce tasks in production environments, such as Google, can be bursty [2][1]. These bursty workload characteristics complicate the challenges of optimizing the performance in a sustainable fashion.

[1]The Google trace is dominated by the MapReduce applications but the applications related details, such as the attributes of map and reduce tasks are not provide.

Executing MapReduce workloads indeed requires a large amount of computation, e.g., CPU and memory, and communication resources, e.g., links and switches. Today's servers and network switches are designed with multiple operating modes, i.e., *on*, *idle* and *sleep* [3], each of which consumes different power and causes various delays to the task execution. Strategies for achieving green datacenter tend to optimize either energy consumption of servers [4] or network [5], by leveraging power saving options for individual hardware or intelligent workload execution for the entire cluster. A few recent studies [6], [7] that control both server and network try to execute workloads by employing as few number of servers as possible, overlooking the energy saving from *sleep* modes and the degradation of response times due to tight packing of tasks. Overall, it is no mean feat to control multiple numbers of resources as well as their operation/power modes for bursty MapReduce workloads without compromising the response time of tasks.

To achieve sustainable and green MapReduce execution in heterogenous data centers, we propose to control MapReduce task assignments in combination with the "sleeping" policy of servers and switches. We leverage the idea of finding the minimal energy path on a datacenter topology, where servers and switches are nodes and links are edges with different capacities and power consumptions depending on operation modes. To such an end, we develop an online GrEen MapReduce Scheduler (GEMS), which assigns incoming tasks to paths that can accommodate their CPU, memory, and external/internal network requirements with minimum energy consumption. GEMS activates sleep mode for servers and switches when possible. In addition to realistic power consumption values on a wide range of heterogenous systems, we employ a task response time model, which includes the delay of switching operation/power modes and the task execution time based on the processor-sharing principle. Essentially, the task response time depends not only on the task requirements but also on the number of co-located tasks.

In particular, GEMS consists of two phases: the first phase finds the minimum energy path that satisfies CPU, memory, and the external traffic requirement of the task and the second phase reserves the bandwidth for paths according to internal all-to-one communication between tasks. Using trace driven simulation, we evaluate GEMS on Google MapReduce trace, one is 4 hour long and the other one is one day long. Our results show that GEMS not only provides significant energy

savings but also leads to a noteworthy decrease in response time of tasks, compared to policies that do not put servers and switches into *sleep* mode, and that aim to only optimize server energy consumption. In a nut shell, GEMS achieves sustainable MapReduce execution by allowing a loose packing of tasks onto a slightly bigger cluster and harvesting the response time reduction due to extra capacity and energy saving by the sleeping mode.

Our contribution can be summarized as follows: *(i)* we employ realistic power consumption values for heterogeneous servers, *(ii)* our response time model considers the impact of executing multiple tasks on the same hardware platform, using the scheduling discipline of processing sharing, *(iii)* the proposed GEMS controls the number of *on* servers and switches as well as their operation mode, *(iv)* our evaluation on traces obtained from production systems show promising gains on energy as well as response time of tasks.

The outline of this work is as follows. In Section II, we explain the details of Google MapReduce workloads and the heterogenous data centers. GEMS is described in Section III. The evaluation results are summarized in Section IV. Section V presents related work, followed by the concluding remark in Section VI.

II. System Model

In this section, we elaborate the complexity of MapReduce workloads, heterogeneous data centers, and different operation/power states of servers as well as switches.

A. The MapReduce Workloads

To illustrate the workload characteristics and resource requirements of MapReduce applications, we use one day long Google cluster trace [2]. The detailed analysis of the trace can be found in [8]. We note that this trace is dominated by MapReduce cluster, meanwhile contain some other applications too. Due to lack of information provided in the trace, it is not possible to differentiate the applications nor the details regarding to MapReduce applications, such as the number of map and reduce tasks. Figure 1(a) summarizes the number of MapReduce tasks arriving in a second over four hours. One can see that the number of arriving tasks is rather low ($<$100) for most of the time but periodically tasks arrive in big bursts ($>$10.000 tasks). Google trace specifies tasks' CPU, memory requirements, and the corresponding execution time. All task requirements and machine resource capacities are normalized with a range between $[0,1]$ according to the machine with maximum capacity. However, the network requirement of tasks is not provided in the trace. In order to obtain the network requirement and the communication patterns of tasks, We use Benson et. al.'s [9] real data center network analysis.

B. Network Requirement

In our system, communication demand of a task is twofold; communication with the master task, and communication with external sources. As for the communication pattern of internal traffic, we assume the tasks of the same job arrive at the beginning of the same slot and one of them is the master task. All tasks belong to same job needs to communicate with master task until task finishes. Bandwidth reservation for internal traffic is made from server which task is assigned to the server which master task is running. So there is an all to one communication for internal traffic. On the other hand, a task needs bandwidth reservation to communicate to an external source during its execution. Bandwidth reservation for external traffic is made from the core switch to the server which task is assigned.

To approximate the network bandwidth demands of tasks, we leverage the network characterization study in [9], which collected the cumulative distribution of flow sizes on real data centers hosting MapReduce applications, as Google trace data does not contain network related information. We propose to estimate internal and external network bandwidth demands per task as their flow sizes divided by the minimum execution time of a task. Here, we approximate the minimum execution time of a task as the CPU demand of a task divided by the server capacity, which the task is assigned to. We note that exact network bandwidth demands depend on the mapping of task-server, whereas the flow size per task is constant. Essentially, for each task, we first randomly draw two flow sizes from the CDF provided in [9] for internal/external communication and then calculate their bandwidth demands that are used to reserve link capacity.

In summary, tasks can be assigned to a server, only if a server has sufficient CPU and memory capacity as well as a routing path with sufficient link capacity.

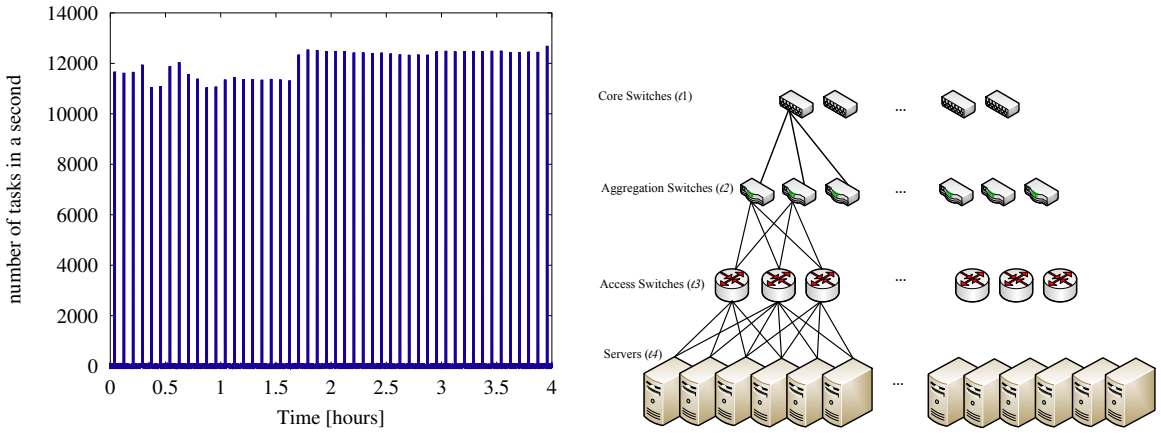
C. Data Center Topology

We consider a typical 3-tier data center architecture where each switch is fully connected with the lower layer nodes by links, as shown in Figure 1(b). The first layer is composed of core switches, denoted as l_1, which can be viewed as the source nodes and where the arriving MapReduce tasks enter the data center. The switch nodes in the next two levels are aggregation and access switches, denoted as l_2 and l_3, respectively. The last node layer is heterogeneous servers, denoted as l_4, which have different CPU, and memory capacities. Node layers are connected through links, which have different bandwidth capacities according to the connected layers.

In particular, the datacenter in our study is equipped with 1250 server nodes, 8 core switches, 16 aggregation switches, and 512 access switches. The link capacities among core, aggregation and access switches are 10 G, 1G and 1G respectively. The specific values regarding to topologies are listed in Table I. Following Google trace data in [2], we consider four types of server nodes, whose respective percentage and CPU/memory capacities are listed in Table II.

D. Energy Consumption

Here, we assume all servers and switches are equipped with the technology supporting three operation modes, i.e., *on*, *idle*, and *sleep*. The differences among those states are the power consumption and transition overhead between states.



(a) MapReduce task arrival patterns

(b) Data center system architecture

Fig. 1. MapReduces and data center

TABLE I
TOPOLOGY: NUMBER OF SWITCHES AND THEIR POWER CONSUMPTION

		Power [watts]		
Node types	Quantity	P_{sleep}	P_{idle}	P_{on}
Core (l_1)	8	5	16.6	25
Aggregation (l_2)	16	10.2	34	51
Access (l_3)	512	15	50	75

TABLE II
CAPACITIES AND POWER CONSUMPTION OF FOUR TYPES OF SERVERS

		Normalized		Power [watts]				
	Ratio	CPU	Memory	$P_{(cpu)}$	$P_{(mem)}$	P_{sleep}	P_{idle}	P_{peak}
A	54%	0.5	0.5	51.5	18	46.25	162	231.25
B	31%	0.5	0.25	51.5	9	44.5	162	222.5
C	8%	0.5	0.75	51.5	24	47.5	162	237.5
D	7%	1.0	1.0	103	36	60.2	162	301

Switching between *idle* and *on* states is automatically triggered by task arrivals and departures and incurs no delay. In contrast to *idle* and *on*, *sleep* mode has much lower consumption but requires (de)activation command. Moreover, a non-negligible overhead in terms of energy and delay can occur when waking up *sleep* servers/switches. Here, we assume extra energy consumption from *sleep* to *on* is computed as the the product of peak power consumption and times duration in such a transition. Moreover, we assume no extra delay and energy consumption are incurred for transition from *on* to *sleep*.

We detail the specific energy consumption of servers and switches in the following subsections. Throughout this paper, we adopt the notation P_s^i as the power consumption at state s, i.e., *on*, *idle*, and *sleep*, for node type i, e.g., servers or switches.

1) Server energy consumption: We model the power consumption of an *on* server with a fixed term, i.e., power consumption at the *idle* state, and load-dependent term that increases proportionally with the memory and CPU usage [10] as the following equation:

$$P_{on} = P_{idle} + P_{(cpu)}U_{cpu} + P_{(mem)}U_{mem} \quad (1)$$

$P_{(cpu)}$ and $P_{(mem)}$ are the power values when CPU and memory are fully utilized. U_{cpu} and U_{mem} denote the utilization of CPU and memory, respectively. U_{cpu} and U_{mem} values are between 0 and 1.

The peak power consumption of an *on* server equals to:

$$P_{peak} = P_{idle} + P_{(cpu)} + P_{(mem)}$$

The energy consumption of server essentially needs to integrate the power consumption over time.

2) Network energy consumption: Power consumption of a switch directly correlates with the number of ports and line cards [11], [12]. Particularly, each type of switch considered here has a fixed number of ports and line cards. As such, we include power consumption of links to the power consumption of the switches to which they are attached. Taking reference values of the power consumption of switches from [13], we consider power values of core, aggregation, access switches as listed in Table I. The overall energy consumption of data center network sums over all the switches.

E. Response Time Model

In addition to the detailed model on energy consumption, we also model the task response time defined as time between task arrival and departure. Response time of a task, $T_{response}$ is composed of two terms: T_{wait} waiting time of task before processing starts, and $T_{execution}$ execution time of the task.

T_{wait} depends on scheduling algorithms and wake up delay of servers. In this study, a task is scheduled immediately after its arrival and servers are woken up if needed. We assume wake-up time of servers and switches are 30 sec and 0 sec, respectively. In a special case where tasks are not allowed to queued at servers that do not have enough capacities upon task arrivals, T_{wait} is limited to the 30 seconds. In this study, we only consider algorithms under this special case.

The second contributor of $T_{response}$ is the execution time of the task, $T_{execution}$. To accommodate the impact of executing multiple tasks on a server at the same time, we propose to scale the execution time of tasks specified in the trace, by the ratio between tasks' CPU requirements and available CPU capacity of server. In particular, we assume the processor sharing principle is used by servers, i.e., co-executed tasks equally share the available capacity.

We illustrate our model by two simple examples. When a task with a CPU requirement of 0.5 is assigned to a server with CPU capacity of 1, its execution time is therefore 1/0.5=2 times faster than the value specified in the trace. When such a task is assigned to CPU of 1 together with another task with CPU requirement of 0.5, it gets only half of the capacity, i.e., $1/2$, and therefore its execution time remains the same as trace. The remaining task execution time is updated at the beginning of each slot according to the excess CPU capacity of the assigned server. As a task can only be assigned to a server, with enough capacity to satisfy the requirement, task execution time in our system is bounded by the original execution time specified in the Google trace.

III. Green MapReduce Scheduler

In this paper we propose a novel online algorithm, GEMS (GrEen MapReduce Scheduler), which assigns MapReduce tasks in a heterogeneous data center with an aim to minimize energy consumption as well as response time. To such an end, GEMS finds the least energy consumption routing path whose nodes satisfy task requirements of CPU, memory and links fulfills external and internal communication. GEMS also controls the operation modes of servers and switches, i.e., enter/wake-up *sleep* modes.

GEMS works in a slotted manner and consists of two phases. In the first phase, GEMS determines task's routing path, ρ' and assigns a server to tasks, accommodating CPU and memory requirements and providing link bandwidth for external communication to the outside world. In the second phase, GEMS finds task's routing path, ψ', which reserves bandwidth for internal communication with master tasks, given the task server assignment determined in the first phase. Moreover, GEMS finds path right upon task arrivals and only assign tasks to servers whose capacities are sufficient at the time slot of arrivals. The outline of GEMS is described in Algorithm 1.

Algorithm 1 GEMS (GrEen MapReduce Scheduler)

```
Phase I:
for each time slot t do
    check finished tasks and update residual capacities
    put idle servers and switches to sleep
    add newly arrived tasks to the task queue
    sort tasks in the ascending order according to memory
    for each task in the job queue do
        compute minimum energy path ρ' according to Equation 2
        assign task and update residual capacities of server and links
        wake up servers and switches along ρ'
    end for
    Phase II:
    for each task do
        find minimum energy cost path ψ' to the server which master task is running
        establish path and update residual capacities of links
        wake up switches along ψ'
    end for
end for
```

A. Phase I:

At the beginning of each slot GEMS starts with checking finished tasks. Once the tasks are finished, the reserved CPU, memory, and link bandwidth are immediately freed. When a server or a switch does not serve any task, it is immediately put into *sleep* mode.

The next step is to find the minimum energy path ρ' for all tasks arriving at the beginning of the slot. The tasks are sorted in an ascending order according to their memory requirements. There are two reasons behind: memory is the typical bottleneck in MapReduce applications and this is indeed observed in our trace. The rationale of sorting tasks in an ascending order is to best take advantage of residual capacities of *on* servers and switches. Sorting tasks in a descending order can easily result in situations where remaining capacities of *on* servers are not sufficient to fit big tasks and a higher number of *on* servers are thus required.

This minimum energy path starts from a dummy root node (all core switches are connected to root) and ends in a server. Note that, we do not pre-determine the end point, i.e., task-server assignment, before routing is found. Essentially GEMS selects server and path simultaneously while searching for the path. GEMS goes through all paths ρ in the system which satisfy the resource requirements of the task and calculates their energy cost and selects the path ρ', which has the minimum energy cost. The energy cost calculation for routing path ρ, $E(\rho)$, depends on the states of switches and server nodes along the path, i.e., their operation modes and number of task already in execution. For each node $i \in \rho$, its energy cost is composed of two terms, transition energy cost and power cost of execution task, P^i_{on}. The transition energy cost is essentially the product of transition power, i.e., peak power (P_{peak}), and the delay before the execution starts, T_{wait}. Note that T_{wait} occurs only when servers wake up from *sleep* mode, and T_{wait} equals to zero changing from *idle* to *on* mode. $T_{execution}$ is the estimated execution time of the task on a specific server. $T_{execution}$ depends on the assigned cpu capacity, hence number of tasks running on that server. And, the calculation of P_{on} depends on the utilization, as shown in Equation 1. As the energy cost of node i is shared across co-executed tasks, denoted as N_i, we propose to estimate the energy cost of executing a task on node i as $\frac{P_{peak}T_{wait}+P^i_{on}T_{execution}}{N+1}$. Therefore, one can estimate energy cost of path ρ as:

$$E(\rho) = \sum_{i \in \rho} \frac{P^i_{peak}T^i_{wait} + P^i_{on}T_{execution}}{N_i + 1} \tag{2}$$

Essentially, the minimum energy path not only satisfies tasks' demands of CPU, memory and external communication but also incurs the least amount of energy consumption. When

servers and switches on path ρ' are in the *sleep* mode, GEMS needs to deactivate them and cause an extra delay in the wake-up processes. After finding the minimum energy path for all tasks, GEMS moves to phase II.

B. Phase II:

In phase II, GEMS aims to find the minimal energy path, ψ', that sustains internal communication with the master task. In contrast to phase I, the minimum energy path is established between two pre-specified servers. The energy cost calculation for ψ is very similar to ρ, except energy cost of server part is not considered. When a task is finished, it is removed from the corresponding server and the bandwidth reserved on communication paths ρ' and ψ' is released.

IV. Results

To evaluate the effectiveness of GEMS on a heterogenous datacenter, we use a large scale MapReduce trace by Google [2] and build a trace driven simulator. In particular, we evaluate a datacenter of 1250 server nodes, whose capacities and power consumption are listed in Table II. This datacenter is connected by a 3-tier fat tree like network, whose topology and power consumption are detailed in Table I. We present results of 4 and 24 hour long traces.

The aim of our evaluation is two-fold. First, we seek to show the advantages of activating *sleep* mode of servers and switches for bursty MapReduce workloads. Second, we emphasize the the energy savings and response time reduction that are resulted from considering server and network consumption simultaneously. The metrics of interests are, the average task waiting time (T_{wait}), average task response time ($T_{response}$), system utilization, and, most importantly, energy consumption of entire system (E_{total}), server part (E_{server}) and network part ($E_{network}$), measured in kWh. The summary of evaluation results is presented in Table III and discussions are detailed in the following subsections.

A. Comparing with noSleep policy

To evaluate the pros and cons of using *sleep* mode in GEMS, we implement a "noSleep" version for GEMS algorithm, i.e., GEMS(noSleep). The difference between GEMS and GEMS(noSleep) is, when a switch or server is not serving any task then it enters immediately into *idle* mode rather than *sleep* mode. As a result, GEMS(noSleep) incurs no waiting time T_{wait} or energy cost for transition from *idle* to *on* mode. Hence, the computation of minimum energy path, $E(\rho)$ and $E(\psi)$, are different in both cases. Nevertheless, *sleep* mode has much lower power consumption than *idle*, for all server and switch types considered in Table II and Table I.

As shown results in Table III, GEMS indeed results into a much lower energy consumption, in terms of total , server and network, compared to GEMS(noSleep). And, GEMS incurs a non-negligible delay, $T_{wait} = 21$ sec, due to waking up *sleep* servers. Surprisingly, GEMS is able to achieve lower task response times than GEMS(noSleep). To further contrast GEMS and GEMS(noSleep), we also summarize the percentages of energy savings and of reduction in task response times in Figure 2(a). Clearly, GEMS has nearly 30% server and 60% network energy savings via adopting *sleep* mode for servers and switches. We reason the prominent energy saving from sleeping switches is due to the bursty workload. As for average task response time, GEMS achieves 124 second per task, accounting for 30% reduction, even though there is an extra delay due to waking up *sleep* servers.

B. Comparing with noNetwork policy

To address the importance of optimizing server as well network energy for MapReduce workloads, we compare GEMS with a green schedule that only tries to optimize the server energy, called GEMS(noNetwork). For a fair comparison, GEMS(noNetwork) also uses same sleeping policy as GEMS, i.e., servers and switches enter *sleep* mode right after completing tasks. GEMS(noNetwork) assigns tasks to servers, by only considering their CPU/memory capacities and the energy cost of servers, i.e., only the destination node of minimum energy path, ρ. As for the task external and internal communication, GEMS(noNetwork) randomly finds internal and external communication paths.

From table III, GEMS(noNetwork) indeed consumes slightly less server energy than GEMS, however consumes much higher network energy. As a net result, GEMS still consumes less total energy than GEMS(noNetwork), i.e., roughly 11%, shown in Figure 2(a). However, the average task response time of GEMS(noNetwork) for 4 hour trace is 207 second, that is roughly 60% higher than the GEMS result, 124 second. Such an observation is due to that GEMS(noNetwork) tries to assigns tasks more tightly into servers. Consequently, the execution times of tasks increase tremendously and only little energy saving from the server part is achieved. On the contrary, GEMS carefully chooses communication path and assign a fewer number of task into servers. GEMS results into shorter task execution times, which in turn bring advantages in saving energy. We note that such a benefit is often overlooked in related studies that do not model task response times.

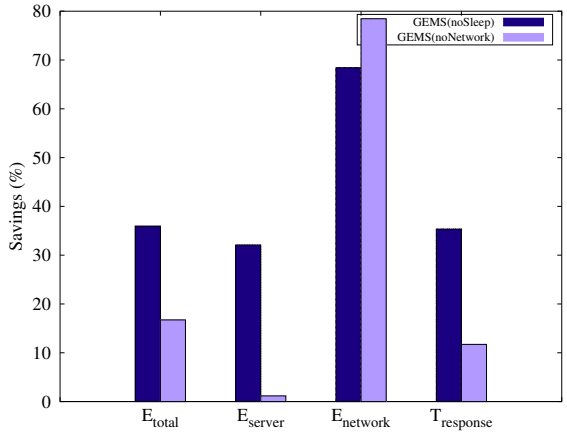
C. 24 Hour Trace

To see the scalability of GEMS in a larger scale, we evaluate GEMS against GEMS(noSleep) and GEMS(noNetwork) under a 24 hour trace. Similar to 4 hour results, GEMS achieves significant energy savings as well as response time reduction.

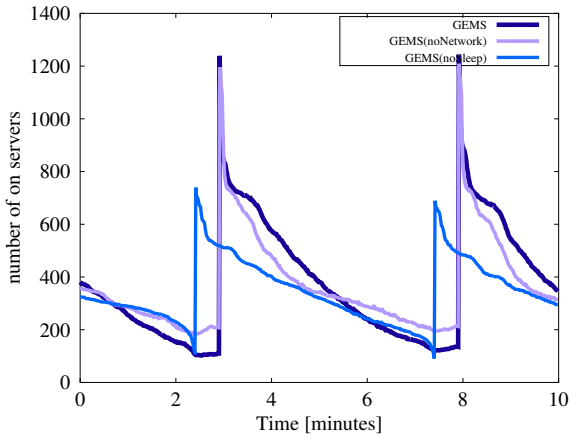
Figure 2(b) shows the number of *on* servers changing over time with all three algorithms. One can see that as GEMS(noNetwork) and GEMS apply sleeping, their response to bursty workload is shifted and appears roughly 30 sec after the arrival of the burst, because of the 30 sec wake up time of servers. Moreover, GEMS is more sensitive to bursty workloads and activates many more new servers. Due to the high number of server provisioning during the burst, tasks are completed faster and therefore the number of *on* servers can drop dramatically after the arrival of a burst. Overall, GEMS is able to provide a higher number of servers for the bursty workloads promptly and effectively sleep servers and switches

TABLE III
RESULTS

	4 HOURS			24 HOURS		
metrics	GEMS	GEMS(noNetwork)	GEMS(noSleep)	GEMS	GEMS(noNetwork)	GEMS(noSleep)
E_{total} [kWh]	806	968	1258	4338	4383	6428
E_{server} [kWh]	764	773	1125	4093	4181	5762
$E_{network}$ [kWh]	42	195	133	245	201	665
T_{wait} [sec]	18	24	0	21	28	0
$T_{response}$ [sec]	128	145	198	138	183	223
average U_{mem} [%]	56	64	63	65	70	66
average U_{cpu} [%]	68	75	78	74	80	82
average *on* servers	340 (max 1248)	341 (max 1224)	316 (max 850)	460 (max 1250)	460 (max 939)	376 (max 943)
average *on* switches	12 (max 13)	533 (max 536)	15 (max 17)	33 (max 48)	2 (max 512)	15 (max 17)



(a) GEMS savings compared against GEMS(noSleep) and GEMS(noNetwork)



(b) number of *on* servers: GEMS vs GEMS(noSleep) vs GEMS(noNetwork)

Fig. 2. Executing MapReduce on a heterogeneous cluster of 1250 nodes

during the low loads, so that both response times and energy consumption are minimized.

V. Related Work

Energy efficiency of data centers is addressed by studies which are looking from server and network point of view. For the server part, dynamic right-sizing is extensively studied using analytical approaches [4], [14]. Dynamic right-sizing studies, optimize the number of *on* servers at a time according to incoming tasks. These studies focus on instantaneous state of the system, and optimizes power consumption by finding minimum number of server to satisfy the load. From the network energy point of view, ElasticTree [5] finds the minimum number of active network elements while satisfying QoS requirements.

Furthermore, there are some recent studies on VM placement while considering cost of communicating tasks. Interest in energy efficient VM placement is increasing as the cloud computing becoming popular. In [6], [7], [15], they all seek to make traffic aware VM placement. However, their objective is to minimize the energy consumption of the network part only. In addition, they consolidate traffic flows rather than making bandwidth reservations.

Our study differs from this literature by providing a holistic energy and response time consideration on a heterogeneous server environment. Our aim is to design an online green scheduler, which minimizes the energy consumption while also taking response time into account. Moreover, we consider time aspect in our analysis, since GEMS minimizes energy rather than power consumption at an instantaneous time moment. Furthermore, we introduce a more realistic data center environment with heterogeneous servers and switches in our analysis.

VI. Summary and Conclusion

Motivated by significant energy draw of MapReduce workloads on today's datacenter, we develop GEMS, a green MapReduce scheduler. GEMS aims to minimize energy consumption of servers as well as network by finding minimum energy path for tasks according to their CPU, memory and network requirements and adopting the *sleep* mode. Overall, GEMS tries to provide a sufficiently high number of servers for the bursty workloads and effectively sleep servers and

switches during the low loads. Using on-production Google MapReduce traces, our simulation results on a heterogeneous cluster of 1250 servers show that GEMS is able to achieve significant energy saving while reducing average task response times, compared to schedulers that overlook sleeping policy or network energy. The efficiency of GEMS lies at promptly providing a higher number of servers during the peak loads, thus leading to a significant reduction on the response time of tasks as well as the cluster energy consumption. For our future work, we plan to extend GEMS to different datacenter topologies and scenarios where tasks have more complex communication dependency.

REFERENCES

[1] J. Dean and L. Barroso, "The tail at scale," *Communications of ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[2] "Google Inc., Efficient Data Center Summit, April 2009." [Online]. Available: http://www.google.com/corporate/green/ datacenters/summit.html

[3] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: Eliminating server idle power," *SIGARCH Comput. Archit. News*, vol. 37, no. 1, pp. 205–216, 2009.

[4] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *INFOCOM 2011*, 2011, pp. 1098–1106.

[5] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: saving energy in data center networks," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, 2010, pp. 17–17.

[6] D. Huang, D. Yang, H. Zhang, and L. Wu, "Energy-aware virtual machine placement in data centers," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, 2012, pp. 3243–3249.

[7] L. Wang, F. Zhang, A. V. Vasilakos, C. Hou, and Z. Liu, "Joint virtual machine assignment and traffic engineering for green data center networks," in *GreenMetrics, 2013 Proceedings IEEE*, 2013.

[8] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale:Google trace analysis," Intel science and technology center for cloud computing, Carnegie Mellon University, Tech. Rep., April 2012.

[9] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, New York, NY, USA, 2010, pp. 267–280.

[10] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, Berkeley, CA, USA, 2008, pp. 337–350.

[11] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, "Energy aware network operations," in *Proceedings of the 28th IEEE International Conference on Computer Communications Workshops*, ser. INFOCOM'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 25–30.

[12] L. J. Wobker, "Power consumption in high-end routing systems," March, 2014. [Online]. Available: www.nanog.org/meetings/nanog54/presentations/Wednesday/Wobker.pdf

[13] D. Kliazovich, P. Bouvry, and S. U. Khan, "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers," *The Journal of Supercomputing*, Nov. 2010.

[14] A. Gandhi, V. Gupta, M. Harchol-balter, and M. A. Kozuch, "Optimality Analysis of Energy-Performance Trade-off for Server Farm Management," *Performance Evaluation*, pp. 1–23, 2010.

[15] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Computer Networks*, vol. 57, no. 1, pp. 179 – 196, 2013.