A Constraint Programming Based Energy Aware Resource Management Middleware for Clouds Processing MapReduce Jobs with Deadlines

Adam Gregory
Dept. of Systems and Computer Engineering
Carleton University
Ottawa, ON, Canada
adamgregory@sce.carleton.ca

Shikharesh Majumdar
Dept. of Systems and Computer Engineering
Carleton University
Ottawa, ON, Canada
majumdar@sce.carleton.ca

ABSTRACT
This paper concerns guarantees on system performance through Service Level Agreement (SLA) compliance and focuses on devising energy aware resource management techniques based on Dynamic Voltage and Frequency Scaling (DVFS) used by resource management middleware in clouds that handle MapReduce jobs. This research formulates the resource management problem as an optimization problem using Constraint Programming (CP). Experimental results presented in the paper demonstrate the effectiveness of the technique.

General Terms
Design; Performance; Measurement; Experimentation.

Keywords
Resource management on clouds; MapReduce with deadlines; Constraint Programming; Energy management; Big data

1. INTRODUCTION
Cloud computing services that deploy a large amount of computing resources to satisfy on-demand requests from users around the world consume significant energy. In 2010, global data centers accounted for between 1.1% and 1.5% of total energy use worldwide and this amount is expected to grow as cloud computing technologies continue to garner increasing interest from researchers and practitioners in both academia and industry [8]. Since energy costs account for an estimated 41.6% of large-scale data center operation costs [4], in order to stay competitive, cloud service providers must develop energy efficient resource management strategies which do not sacrifice service quality.

Energy aware resource management techniques are especially important to cloud data centers which provide services for incredibly large and complex data sets. These big data systems handle volume, velocity, variety, and veracity of application data and require a substantial number of computational resources. Reducing costs due to energy consumption within data centers would significantly impact the ability of the cloud service providers to provide the infrastructure for practitioners in both industry and academia to perform big data analytics.

ICPE'16 Companion, March 12-18, 2016, Delft, Netherlands

DOI: http://dx.doi.org/10.1145/2859889.2859892

This research focuses on performance optimization and analysis of resource management middleware that has two primary objectives: ensuring quality of service requirements are satisfied and minimizing energy consumption. In this paper we propose an energy-conscious resource management approach for a batch of requests which require multiple stages of execution and are characterized by a Service Level Agreement (SLA) that comprises of an earliest start time, execution time, and (soft) deadline specified by the user. In systems with such soft deadlines, requests are sometimes permitted to miss their specified deadlines. However, minimizing the number of missed deadlines is important for the overall quality of service. Our approach makes use of available slack time in the execution window of a request by applying an energy-saving CPU frequency reduction-based technique during the execution of some tasks. Thus energy consumption can be reduced without sacrificing service quality.

The workloads considered in this paper consist of a batch of MapReduce jobs. MapReduce is a programming model proposed by Google for processing large amounts of data in a distributed manner [2]. Jobs in the MapReduce programming model are characterized by two stages of execution: a map stage, and a reduce stage. The map stage performs filtering and sorting to generate an intermediate set of key/value pairs. Each task in the map stage processes a small portion of the much larger input data. The generated key/value pairs are then merged in the reduce stage where an application specific summary operation is performed. Input, output, and intermediary data is stored on a distributed file system. A typical MapReduce job consists of a set of map tasks and a set of reduce tasks which do not generally begin execution until all map tasks are complete.

Our research formulates the complex energy aware resource management problem as an optimization problem. Tasks must be allocated to resources in a process known as matchmaking and assigned a scheduled start time. We solve this problem in a joint step using a Constraint Programming (CP) based approach. CP [10] is a well-known programming paradigm intended to find an optimal value with respect to minimizing or maximizing a specified objective function. CP is a form of declarative programming wherein properties of the solution to be found are specified as inviolable constraints. Values are assigned to the set of discrete decision variables such that the objective function is optimized (maximized or minimized depending on the objective).

The main contributions of this paper include:

- A CP-based resource management technique for minimizing energy consumption of clouds executing workloads comprised of a batch of MapReduce jobs subject to SLAs,
- A preliminary performance analysis of this technique that demonstrates its superiority over existing resource management approaches and provides insights into system performance.

The remainder of the paper is structured as follows. Section 2 presents related work. Section 3 describes the energy, workload, and system models. The proposed technique is described in Section 4. A preliminary performance evaluation is presented in Section 5. Finally, conclusions and future work are given in Section 6.

2. PRIOR RESEARCH

There has been a significant amount of prior research on reducing energy consumption in large-scale data centers. In [1] and [15], researchers propose collocating tasks with complementary resource requirements and execution time in order to consolidate workload on fewer servers so that idle servers can be powered down. Wirtz and Ge [14] studied the effect of dynamically scaling processor voltage and frequency to scale system performance and energy consumption based on load. However, these approaches do not consider tasks characterized by SLAs.

Researchers in [7] and [3] considered the tradeoff between task deadline compliance and energy consumption and found that energy consumption could be reduced without significant degradation to deadline compliance. The workloads investigated in these studies included only jobs which require a single stage of execution and did not consider precedence relationships between tasks.

Verma et al. [12] proposed a heuristic scheduler which uses two novel resource allocation policies based on Earliest Deadline First (EDF) for scheduling MapReduce tasks with deadlines. In [9], Lim et al. devised a technique for determining an optimal schedule for a batch of MapReduce jobs using Mixed Integer Linear Programming (MILP) and Constraint Programming (CP) which minimizes the number of tasks which miss their deadlines. These studies did not consider energy consumption which is the focus of attention in this paper.

3. MODELS

3.1 Energy Model

We adopt the energy model from [13]: host power $P = P_s + P_d$, where P_s and P_d are static and dynamic power of the host. Static power is primarily architecture and hardware dependent whereas dynamic power depends on load. Based on previous research we assume $P_s \propto P_d$. Thus, the energy required to execute a given program can be expressed as:

$$P = \delta * V^2 * f \quad (1)$$

$$E = \sum_{\Delta t} P * \Delta t \quad (2)$$

Where Δt is the duration the program is executed at a given operating frequency f with processor source voltage V and δ is a constant based on hardware characteristics.

The Advanced Configuration and Power Interface (ACPI) specification provides an open standard that operating systems can use to manage power [5]. An ACPI-enabled host supports a number of discrete voltage/frequency configurations and can change its configuration while computation is ongoing using Dynamic Voltage and Frequency Scaling (DVFS). We denote f^{high} and f^{low} as the maximum and minimum permitted operating frequencies respectively. Because frequency scaling is instantaneous, any desired average operating frequency between f^{high} and f^{low} is possible by varying the proportion of time permitted frequencies are used. In multicore processors, voltage and frequency of each core can be scaled independently.

ACPI-enabled hosts have several power modes collectively called C-states [5]. In this paper we adopt C1 (commonly known as Halt) when a processor core is unused and permitted to enter a sleep state. A halted processor core can be returned to the operating state with negligible overhead. Power consumption in state C1 is assumed to be a percentage, defined as *halt percentage*, h, of the power consumed in operating state C0 when the core is running at the maximum permitted frequency f^{high}.

3.2 Workload Model

Similar to prior research [9], the input workload comprises a batch of n MapReduce jobs which are represented by the set $J=\{j_1,j_2,\ldots,j_n\}$. Each job, j, in J has the following characteristics: a deadline d_j by which the job should be completed, an earliest start time s_j, a set of map tasks $T_j^{map} = \{t_{j,1}^{map}, t_{j,2}^{map}, \ldots, t_{j,k_j^{map}}^{map}\}$ where k_j^{map} is the total number of map tasks in job j, a set of reduce tasks $T_j^{red} = \{t_{j,1}^{red}, t_{j,2}^{red}, \ldots, t_{j,k_j^{red}}^{red}\}$ where k_j^{red} is the total number of reduce tasks in job j, and a set $T_j=\{T_j^{map}, T_j^{red}\}$ which contains all of the tasks for job j.

Each task in the set T_j has an execution time e_t. Execution time of a task is defined as the completion time when executed at f^{high}. The tasks of all jobs are stored in the set $T=\{T_1,T_2,\ldots,T_n\}$.

Each, workload investigated has an associated *missed deadline ratio*, d, and a *laxity factor*, l. Missed deadline ratio specifies the maximum proportion of jobs in the set J permitted by the client to miss their soft deadlines. Laxity factor specifies the amount of available slack time in the execution window of jobs in the workload and is used to generate the deadline of all jobs. Laxity factor is defined as one plus the ratio of laxity of job j to the sum of execution times of all tasks in j where job laxity is defined as:

$$Laxity\ (job\ j) = d_j - \sum_{t \in T_j} e_t - s_j \quad (3)$$

Workloads with a higher laxity factor allow for a greater degree of freedom in the scheduling of tasks due to relaxed job deadlines.

3.3 System Model

Each task defined in the input workload is mapped and scheduled onto one of m cloud resources in the set $R = \{r_1,r_2,\ldots,r_m\}$. The model for each cloud resource r_i has: (1) a map task capacity (number of map slots) c_r^{map}, (2) a reduce task capacity (number of reduce slots) c_r^{red}, and (3) a processor P. Parameters c_r^{map} and c_r^{red} specify respectively the maximum number of map and reduce tasks which can be executed in parallel on the resource at any given point in time. Each task slot can be modeled as an independent core in the processor P which has: (1) a discrete set of frequency-voltage pairs which lists the valid processor operating frequencies and corresponding supply voltages and (2) a halt percentage, h, which defines the power consumed while idle as a percentage of the power consumed while operating at f^{high}. These parameters are used to determine the amount of energy consumed by each resource slot when executing tasks or idling. In line with other research in this area [13], this paper assumes a homogenous cloud environment with the same processor P used to model the energy consumption of each of the cloud resources in R.

4. PROPOSED TECHNIQUE

This section describes the energy aware DVFS-based resource management approach. The CP model objective function and constraints are presented in Table 1.

The CP model has the following decision variables:

- A binary variable x_{tr}, which is set to 1 if task t is assigned to resource r and set to 0 otherwise. There is an x_{tr} variable for each combination of tasks in T and resources in R.

- An integer variable a_t, which specifies the scheduled start time for task *t*. There is an a_t variable for each task in *T*.
- An integer variable c_t, which specifies the execution duration for task *t*. c_t determines the amount of frequency scaling applied to the execution of task *t*. There is a c_t variable for each task in *T*.
- A binary variable N_j, which is set to 1 if job j misses its deadline and set to 0 otherwise. There is an N_j variable for each job in *J*.

Table 1. CP Model: Objective Function and Constraints (con)

$$minimize \sum_{t \in T} P\left(\frac{e_t}{c_t} f^{high}\right) * c_t + h * P(f^{high}) * \left(\sum_{r \in R} \left(\max_{t \in T}(a_t + c_t) * (c_r^{map} + c_r^{red})\right) - \sum_{t \in T} x_{tr} * c_t\right) subject\ to$$

$$\sum_{r \in R} x_{tr} = 1 \; \forall \, t \in T \quad \text{(con1)}$$

$$\left(a_t \geq s_j \; \forall \, t \in T_j^{map}\right) \forall \, j \in J \quad \text{(con2)}$$

$$\left(a_{t`} \geq \max_{t \in T_j^{map}}(a_t + c_t) \; \forall \, t` \in T_j^{red}\right) \forall \, j \in J \quad \text{(con3)}$$

$$\left(\max_{t \in T_j^{red}}(a_t + c_t) > d_j \implies N_j = 1\right) \forall \, j \in J \quad \text{(con4)}$$

$$(cumulative((a_t | x_{tr} = 1), (c_t | x_{tr} = 1), 1, c_r^{map}) \forall \, t \in T_r^{map})) \forall r \in R \quad \text{(con5)}$$

$$(cumulative((a_t | x_{tr} = 1), (c_t | x_{tr} = 1), 1, c_r^{red}) \forall \, t \in T_r^{red})) \forall r \in R \quad \text{(con6)}$$

$$\sum_{j \in J} N_j \leq |J| * d \quad \text{(con7)}$$

$$(x_{tr} \in \{0,1\} \forall \, t \in T) \; \forall \, r \in R \quad \text{(con8)}$$

$$N_j \in \{0,1\} \; \forall \, j \in J \quad \text{(con9)}$$

$$\left(c_t \in \mathbb{Z} \mid e_t \leq c_t \leq e_t \frac{f^{high}}{f_{low}}\right) \forall \, t \in T \quad \text{(con10)}$$

$$a_t \in \mathbb{Z} \; \forall t \in T \quad \text{(con11)}$$

Explanation of Constraints: the input of the CP model comprises of a set of jobs *J* which are executed on a set of resources *R*. Each task must be mapped to exactly one cloud resource (con1). Each task belonging to job *j* can only begin at or after job *j*'s earliest start time, s_j (con2). All map tasks in the set T_j^{map} must be completed before reduce tasks in the set T_j^{red} can begin execution (con3). Finally, the capacity limits of each resource in *R* cannot be violated at any point in time (con5 and con6).

Constraints (con1) to (con6) adapted from [9] (modified to suit current research requirements) ensure that these general matchmaking requirements are satisfied and the resulting output schedule is valid. con7 guarantees that the output schedule meets the client specified grade of service by limiting the number of jobs which are permitted to miss their deadlines to less than the missed deadline ratio, *d*. The remaining constraints, (con8) to (con11), define the domains of the decision variables.

con5 and con6 are expressed using the global constraint cumulative. The cumulative constraint requires two input parameters: a collection of tasks and a limit. Each task in the collection has an associated start time, duration, and resource requirement. The cumulative constraint ensures that the sum of the resource requirements of tasks in the collection which have an overlap in their scheduled execution do not exceed the specified limit at any point in time. The resource requirement of each task is set to one because each task executes on exactly one slot.

Explanation of objective function: the objective of the CP implementation is to generate an output schedule, which includes the allocation of tasks to resources, scheduled start time of tasks, and scheduled end time of tasks, that minimizes the energy required to execute a batch of MapReduce jobs. The calculation of energy consumption, shown in the CP model formulation objective function (see top of Table 1), is broken down into two components: Task Energy and Idle Energy.

Task Energy is the energy consumed by resources to execute tasks in the workload. The energy required to execute a given task can be expressed as $E_t = P\left(\frac{e_t}{c_t} f^{high}\right) * c_t$ where *P(x)* is a function that returns the power when the processor operates at frequency x.

Idle Energy is the component of total energy consumption due to processor cores in halt state C1 during workload execution. Recall that power consumption in this state is dependent on halt percentage, h, and can be expressed as $P_{C1} = h * P\left(f^{high}\right)$. The duration a given resource is in state C1 can be calculated as the batch completion time multiplied by the number of cores minus the total execution time of all tasks executed on that resource.

The CP model is expressed in Java using IBM ILOG CPLEX Concert Technology and solved using the CPLEX CP Optimizer Java API [6].

5. PERFORMANCE EVALUATION

Performed experiments simulate the scheduling and execution of a batch workload on a closed system to evaluate the performance achieved by the energy aware MapReduce resource manager. Each experiment concluded after all workload tasks were successfully scheduled and the output schedule was generated. Workload execution was simulated using the output schedule to determine batch completion time and energy consumption.

The performance of the proposed energy aware resource manager (System II) was compared against "Approach 3" (System I) from [9], a resource management approach for batch workloads subject to SLAs that include soft deadlines with the primary objective of minimizing the number of missed deadlines, based on the following metrics:

- *Energy consumption (E)*: total energy consumed to execute the batch workload.
- *Batch Completion time (C)*: time at which the last task in the batch workload finishes execution.
- Number of missed deadlines *(N)*

Note that due to the size and complexity of the matchmaking and scheduling problem, the CPLEX CP Optimizer is unable to prove that a solution is optimal in a reasonable amount of time. Execution time of the CPLEX CP Optimizer is limited to 1% of the total execution time of all tasks in the workload. The output schedule is not guaranteed to be optimal but is the best solution found within the limited time. Experiments with higher time limits yielded negligible improvement in system performance.

5.1 Workload Description

Table 2 presents the system and workload parameters used to compare the performance of System I and System II. This synthetic workload is similar to those used by other researchers and is adapted from [9].

The earliest start time, map task execution times, and reduce task execution times for each job are generated using a discrete uniform distribution. Job deadlines are calculated as the sum of execution times of all the tasks in the job (e_j^{tot}) multiplied by laxity factor, *l* plus earliest start time. The ceiling function is used to ensure that

the resulting deadline is an integer as required by the algorithm implementation.

Table 2. Workload and System Parameters

	Jobs (s_j *and* d_j in seconds)	Task execution times (in seconds)	Resources
medium	n = 10: $s_j \sim$DU(1,50) $d_j = \lceil s_j + e_j^{tot} * l \rceil$ $k_j^{map} = 10$ $k_j^{red} = 5$	$e_t^{map} \sim$ DU(1,25) $e_t^{red} \sim$ DU(1,75)	m = 15: $c_t^{map} = 2$ $c_t^{red} = 2$
small	n = 5: $s_j \sim$DU(1,50) $d_j = \lceil s_j + e_j^{tot} * l \rceil$ $k_j^{map} = 10$ $k_j^{red} = 3$	$e_t^{map} \sim$ DU(1,15) $e_t^{red} \sim$ DU(1,75)	m = 10: $c_t^{map} = 2$ $c_t^{red} = 2$

The resource management process does not depend significantly on the size of input data. As input data size increases, there is a corresponding increase in either number of map tasks or map task execution time. Matchmaking and scheduling of this larger set of input tasks is performed by the resource manager in the same way. Performance for various input data sizes can be investigated by varying either the mean of the discrete uniform distribution which determines map task execution time or the number of map tasks.

The performance of the energy aware resource manager was evaluated for small and medium scale workloads. Because the energy aware resource manager takes advantage of excess slack in the execution window of jobs to reduce the processor operating frequency during the execution of some tasks it was believed that workloads with a small number of tasks may not benefit substantially from this approach. Investigation of the performance for small workloads was deemed worthwhile for this reason.

5.2 Processor Description

The processor model used in this research is based on the frequency/voltage pairs for the Intel Core i7-2760QM presented in [11]. These operating configurations are shown in Table 3.

Table 3. Intel Core i7-2760QM Operating Configurations

Frequency (MHz)	Voltage (V)	Power (W)
2400	1.060	δ *2696.64
2000	0.970	δ *1881.8
1600	0.890	δ *1267.36
800	0.760	δ *462.08

The power consumed in each frequency/voltage configuration is calculated using equation 1 and shown in the Power column of Table 3. A linear regression of these resulting power values is used to express power as a function of operating frequency:

$$P = \delta * (1.364f - 741.81) \quad (4)$$

The correlation coefficient for this linear regression was found to be $R^2 = 0.9699$.

The power estimate presented in equation 4 is based on empirical measurements of processor supply voltage performed by researchers in [11] and is valid only for frequencies between f^{high} and f^{low}.

5.3 Experimental Results

The experiments were performed on a PC running under Windows 7 Professional with a 3.2GHz Intel Core i5 CPU and 4GB of RAM. Reduced energy consumption for the energy-aware MapReduce resource manager can be expected when running the solver on a PC with a faster CPU and more memory. However, these improvements have been observed to be negligibly small. Each experiment was repeated to generate confidence intervals at a 95% confidence level with intervals within ±5% of the mean.

The experiments use a one-factor-at-a-time approach to observe the impact of varying system parameters on overall performance. Laxity factor and missed deadline ratio are varied independently while other system and workload parameters are held at their default values. Default values for laxity factor, missed deadline ratio, and halt percentage are set to 0.6, 0.1, and 5% respectively. Experiments performed using other combinations of parameters showed similar trends. The energy consumption of System I and System II is expressed as energy savings which is calculated as:

$$E_{savings} = \left(1 - \frac{E_{(for\ SystemII)}}{E_{(for\ SystemI)}}\right) * 100\% \quad (5)$$

5.3.1 Medium Workload

Effect of laxity factor (l): The impact of varying laxity factor, *l*, on batch completion time and energy savings for the medium workload can be seen in Figure 1a and Figure 1b respectively. It can be seen that increasing laxity factor results in an increase in energy savings and the batch completion time for System II.

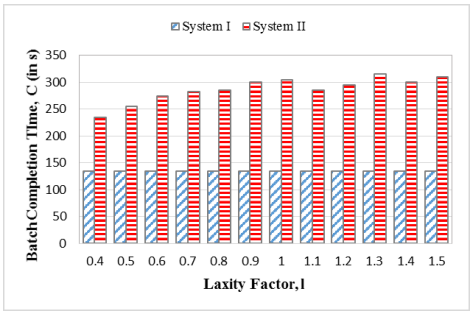

Figure 1a. Impact of Laxity Factor on Batch Completion Time for Medium Workload

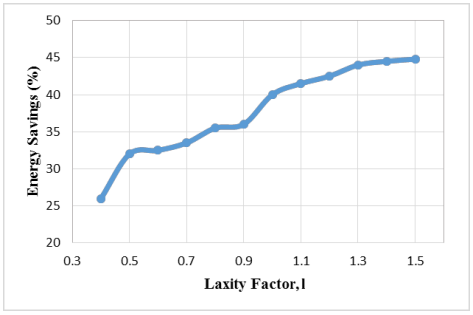

Figure 1b. Impact of Laxity Factor on Energy Savings for Medium Workload

Energy savings and batch completion time for System II do not significantly increase for values of laxity factor above 1. As laxity factor increases, the slack time in the execution window of jobs in the workload increases. System II takes advantage of this slack by reducing execution frequency of some tasks to minimize energy consumption. Additional slack does not further reduce energy consumption or increase batch completion time for System II once all tasks in the workload are executing at the minimum frequency.

Effect of missed deadline ratio (d): The impact of varying missed deadline ratio, *d*, on batch completion time and energy savings for the medium workload can be seen in Figure 2a and Figure 2b respectively. As expected, the results show that as missed deadline ratio increases, batch completion time and energy savings increase.

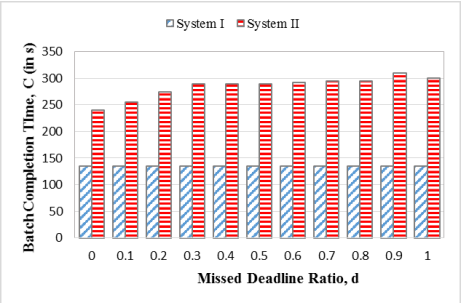

Figure 2a. Impact of Missed Deadline Ratio on Batch Completion Time for Medium Workload

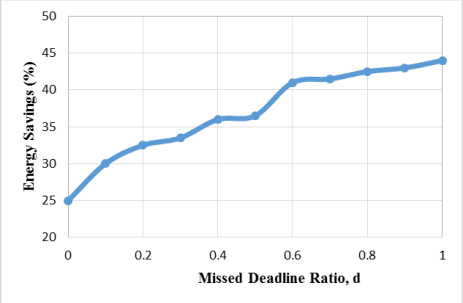

Figure 2b. Impact of Missed Deadline Ratio on Energy Savings for Medium Workload

Although the output schedule for System II results in a later batch completion time in all cases compared to System I this does not indicate a reduction in service quality because the total number of missed deadlines never exceeds the limit specified by the missed deadline ratio. A summary of the tradeoff between increase in batch completion time and energy savings is presented in Table 4. Increase in batch completion time for System II with respect to System I can be expressed as:

$$\Delta C = C_{(for\ SystemII)} - C_{(for\ SystemI)} \quad (6)$$

Table 4. Completion Time and Energy Savings Tradeoff

Increase in Completion Time (s)	110	125	130	145	160
Energy Savings (%)	25	30	33	40	42

System II takes advantage of the relaxation of the grade of service parameter to save energy by reducing execution frequency of some tasks thereby delaying completion of the workload.

5.3.2 Small Workload

Effect of laxity factor (l): The impact of varying laxity factor, *l,* on batch completion time and energy savings for the small workload can be seen in Figure 3a and Figure 3b respectively. A similar trend compared to the medium workload can be observed: increasing laxity factor causes a corresponding increase in both energy savings and batch completion time.

It is interesting to note that the knee of the energy savings graph is observed at laxity factor approximately equal to 0.7 for the small workload compared to approximately 1.0 in the medium workload. After this point, further increasing the laxity factor does not substantially impact either energy savings or batch completion time of System II. It is believed that at this knee point the majority of workload tasks are scheduled to execute at the minimum allowed processor operating frequency and further increasing laxity in job deadlines has only marginal impact on performance. The lower knee point suggests that although fewer resources are used, the resources are more lightly loaded in the small workload experiments compared to in the medium workload experiments.

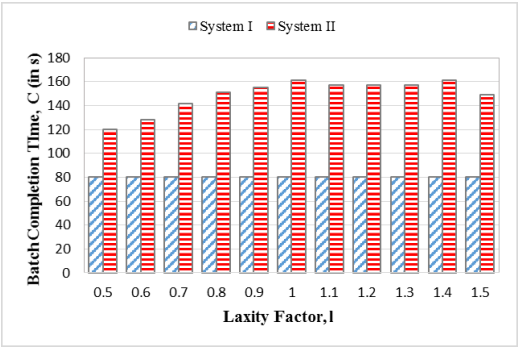

Figure 3a. Impact of Laxity Factor on Batch Completion Time for Small Workload

Additionally, the possible energy savings and relative increase in batch completion time for System II are smaller in the small workload than observed in the medium workload. Because the small workload contains significantly fewer total tasks compared to the medium workload, there are substantially fewer opportunities to save energy by reducing the processor frequency during task execution. Furthermore, because of the lighter workload, idle energy is a higher proportion of total energy consumption. Since idle energy is not reduced using this technique, the total possible energy savings is reduced.

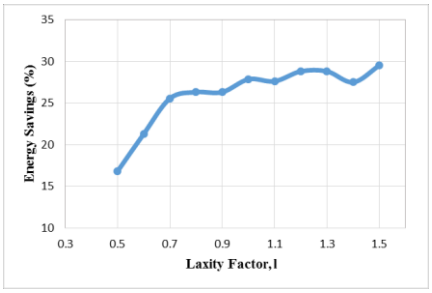

Figure 3b. Impact of Laxity Factor on Energy Savings for Small Workload

A feasible solution could not be found using either System I or System II when laxity factor was set to 0.4. For low values of laxity factor it is possible that the minimum execution time of the longest task in a job is longer than the jobs permitted execution window. In this case the job cannot be scheduled without missing its deadline and, depending on the missed deadline ratio, scheduling the workload may not be possible.

Effect of missed deadline ratio (d): The impact of varying missed deadline ratio, *d,* on batch completion time and energy savings can be seen in Figure 4a and Figure 4b respectively. A similar trend is to the medium workload is observed. A full discussion of these results is not presented due to space considerations.

5.3.3 Number of Missed Deadlines

It should be noted that because the primary objective of System I is to minimize the number of missed deadlines rather than reduce energy consumption, fewer jobs will miss their deadlines under System I than System II. However, in both cases, the number of missed deadlines will never exceed the grade of service parameter missed deadline ratio specified by the client. Furthermore, missed deadline ratio specifies an upper limit to the number of jobs which are permitted to miss their deadlines. In some cases (specifically for high values of laxity factor) fewer jobs than permitted will need to miss their deadlines under System II in order to minimize energy consumption due to excessive slack in job deadlines.

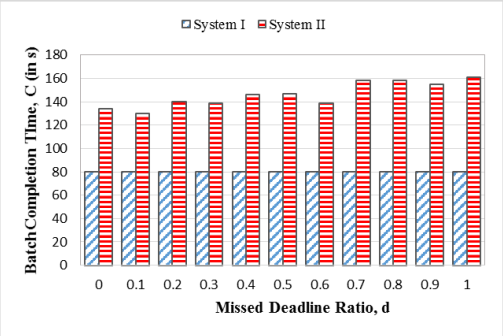

Figure 4a. Impact of Missed Deadline Ratio on Batch Completion Time for Small Workload

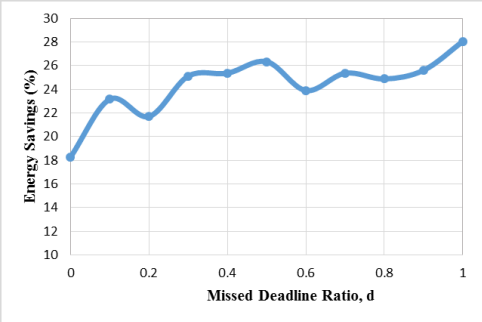

Figure 4b. Impact of Missed Deadline Ratio on Energy Savings for Small Workload

6. SUMMARY AND CONCLUSIONS

This paper introduces an energy aware resource management technique for batch workloads of MapReduce jobs subject to SLAs which include earliest start times, task execution times, and soft deadlines specified by the user. The technique makes use of available slack time in the execution window of jobs and applies a DVFS-based processor frequency reduction to the execution of some tasks to reduce energy consumption *without violating SLAs*. Preliminary performance analysis demonstrates energy savings between 16% and 45% for varying values of missed deadline ratio, laxity factor, and workload size using the DVFS-based approach compared to previous resource management approaches which do not consider energy. Smaller, but still substantial, energy savings were observed for workloads with fewer total tasks. The energy savings were accompanied by small – moderate increases in batch completion times. Plans for future research include:

- Extensively evaluating performance impact of varying workload size and processor model including large workloads from [9] and alternative processor models presented in [11].
- Extending the DVFS-based approach to handle an open stream of job arrivals subject to SLAs.
- Incorporating MapReduce task execution time prediction models to evaluate the impact of and devise techniques to handle error in execution times estimates of MapReduce tasks.
- Enhancing the DVFS-based approach to consider data locality and data transfer times for performance measurements of an implementation on a real-world Hadoop Custer.

7. REFERENCES

[1] Cardosa, M., Singh, A., Pucha, H., Chandra, A., "Exploiting Spatio-temporal Tradeoffs for Energy- Aware MapReduce in the Cloud," *Cloud Computing (CLOUD), 2011 IEEE International Conference on* , vol., no., pp.251,258.

[2] Dean, J. and Ghemawat, S. 2004. MapReduce: Simplified data processing on large clusters. International Symposium on Operating System Design and Implementation (December 2004). 137–150.

[3] Íñigo Goiri, Josep Ll. Berral, J. Oriol Fitó, Ferran Julií, Ramon Nou, Jordi Guitart, Ricard Gavaldí, and Jordi Torres. 2012. Energy-efficient and multifaceted resource management for profit-driven virtualized data centers. Future Gener. Comput. Syst. 28, 5 (May 2012), 718-731

[4] Hamilton, J. "Cooperative expendable micro-slice servers(cems): low cost, low power servers for internet-scale services," in *Proc. of the Conf. on Innovative Data Systems Research,* 2009.

[5] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation, Advanced Configuration and Power Interface Specification, Revision 5.0a, December 6, 2011.

[6] IBM. IBM ILOG CPLEX Optimization Studio. Available: http://www-03.ibm.com/software/products/us/en/ibmilogcpleoptistud

[7] Kyong Hoon Kim, Buyya, R., Jong Kim, "Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters," in Cluster Computing and the Grid, 2007. CCGRID 2007. vol., no., pp.541-548, 14-17 May 2007

[8] Koomey, J. 2011. Growth in data center electricity use 2005 to 2010. Oakland, CA: Analytics Press. August, vol . 1.

[9] Lim, N, Majumdar, S, and Ashwood-Smith, P. 2014. "Engineering Resource Management Middleware for Optimizing the Performance of Clouds Processing MapReduce Jobs with Deadlines", in Proc. 5Th ACM/SPEC ICPE Dublin, March 2014.

[10] Rossi, F, van Beek, P, and Walsh, T. 2004. "Chapter 4: Constraint Programming," in Handbook of Knowledge Representation, San Diego, CA: Elsevier Science. 181-211.

[11] Tan, L and Chen, Z. 2015. Slow down or halt: Saving the optimal energy for scalable HPC systems. *in Proc. ICPE, 2015*, pp. 241–244.

[12] Verma, A.; Cherkasova, L.; Kumar, V.S.; Campbell, R.H., "Deadline-based workload management for MapReduce environments: Pieces of the performance puzzle," *Network Operations and Management Symposium (NOMS), 2012 IEEE* , vol., no., pp.900,905, 16-20 April 2012

[13] Wang, L, von Laszewski, G, Dayal, J, and Wang, F. 2010. "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs," in Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 368–377.

[14] Wirtz, T and Ge, R. 2011. Improving MapReduce energy efficiency for computation intensive workloads. *Green Computing Conference and Workshops (IGCC), 2011 International*, vol., no,pp.1,8, 25-28.

[15] Yanfei Li, Ying Wang, Bo Yin, Lu Guan, "An energy efficient resource management method in virtualized cloud environment," *Network Operations and Management Symposium (APNOMS), 2012 14th Asia-Pacific* , vol., no., pp.1,8, 25-27 Sept. 2012