Cost Efficient Scheduling of Heterogeneous Static Workflow for DVFS Clusters

Weilian Luo
University of Florida
Gainesville, Florida 32603
Email:weilian@cise.ufl.edu

Xun Zhao
University of Florida
Gainesville, Florida 32603
Email:xun.zhao@ufl.edu

Yiwen Zhang
and Biying Fu
University of Florida
Gainesville, Florida 32603
Email:yiwen0707@ufl.edu
Email:fubiying123@ufl.edu

***Abstract*—Energy consumption is one of the largest cost drivers for database centers. As computing demand increases, the machine density of datacenters increases proportionally. Hence, the energy cost also goes up. It is important for companies to minimize energy cost, whether it is for green energy purposes or out of a need for cost savings. This paper studies workflow placement on Dynamic Voltage Frequency Scaling (DVFS) clusters so that energy cost is minimized. A heuristics algorithm will be purposed and early test results justify the value of the algorithm.**

I. Introduction

Nowadays, with the emergence of high end computing and large-scale data centers, energy consumption has become one of the largest cost drivers for database centers. As computing demand increases, the machine density of datacenters increases proportionally. Hence the energy cost also goes up. Many megawatts of electricity are required to power such data centers, and companies like Google and Microsoft spend a large portion of their overall operational costs (e.g., tens of millions of dollars) on electricity bills. It is important for companies to minimize energy, which can also bring benefits such as increasing system reliability as well as reducing operating costs.

Most processors today are using Dynamic Voltage Frequency Scaling (DVFS) technique, which means that the processors can be operated under different voltages and multiple frequencies. DVFS is a commonly-used power-management technique to save power on a wide range of computing systems, from embedded, laptop and desktop systems to high-performance server-class systems, where the clock frequency of a processor is decreased to allow a corresponding reduction in the supply voltage. This reduces power consumption, which can lead to significant reduction in the energy required for a computation, particularly for memory-bound workloads and data intensive workloads. [2]

In our work, we considered the problem of cost efficient scheduling of heterogeneous static workflow for DVFS clusters. We studied workflow placement on DVFS clusters so that the energy consumption of high end computing can be reduced by scaling the supply voltage of processors. Then we proposed a greedy heuristics scheduling algorithm which is able to reduce energy consumption of a large scale cluster of computing resources by using DVFS mechanism. We are trying to finish all incoming jobs with low energy cost clusters, and only using high energy (and thus, higher cost) clusters under two circumstances. The first is when there is no available RAM or CPU at low energy cost clusters for any incoming jobs. The second are when the execution time of a job at low energy cost cluster exceeds its deadline. If so, we move this job to high energy cost cluster if there is a large decrease in job execution time. However, the second circumstance will increase the energy cost so there is a decision making element in the process. Our paper mainly studies this problem in depth. Although theres many algorithms can be used to schedule jobs, but in our paper, greedy algorithm seems like a reasonable algorithm for us to use.

A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimal solution. So a greedy heuristic can give a local optimal solution that approximate a global optimal solution in a reasonable time. Every local optimal choice at greedy algorithm only effect current states, it has no contribution to future states.

In general, greedy algorithms have five components:

1. A candidate set, from which a solution is created.
2. A selection function, which chooses the best candidate to be added to the solution.
3. A feasibility function, that is used to determine if a candidate can be used to contribute to a solution.
4. An objective function, which assigns a value to a solution, or a partial solution.
5. A solution function, which will indicate when we have discovered a complete solution.

Greedy algorithms produce good solutions on some mathematical problems, but not on others. Most problems for which they work will have two properties:

A. Greedy choice property

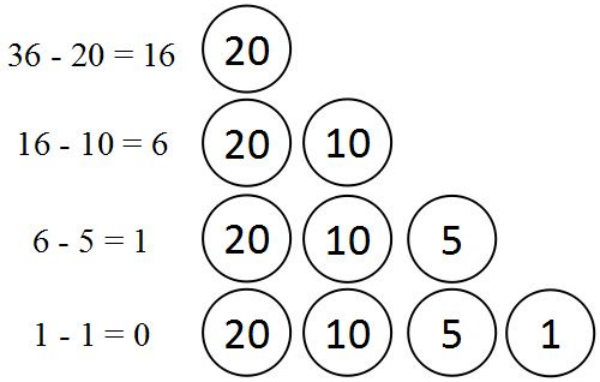
We can make whatever choice seems best at the moment and then solve the sub problems that arise later. The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the sub

problem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices. This is the main difference from dynamic programming, which is exhaustive and is guaranteed to find the solution. After every stage, dynamic programming makes decisions based on all the decisions made in the previous stage, and may reconsider the previous stage's algorithmic path to solution.

B. Optimal substructure

"A problem exhibits optimal substructure if an optimal solution to the problem contains optimal solutions to the sub-problems."

Here is an easiest sample to understand the essence of greedy algorithm:



Greedy algorithms determine minimum number of coins to give while making change. These are the steps a human would take to emulate a greedy algorithm to represent 36 cents using only coins with values 1, 5, 10, 20. The coin of the highest value, less than the remaining change owed, is the local optimal.

With these in mind, the remainder of the paper is organized as follows. Section II discusses related works.Section III describes the definitions of problem. Section IV and V presents formal model definitions and extended model definitions. Section VI is the result of our simulation of MILP and greedy scheduler. Finally, Section VII concludes the paper.

II. Related Works

This section discusses some related works of DVFS, workflow scheduling and cluster computing.

A. Reducing CPU energy cost

In [3], the authors trying to saving energy by reducing CPU energy cost, they consider a new method for reducing the energy used by the CPU. Also, they introduce a new metric for CPU energy performance; millions-of-instructions-per-joule (MIPJ) which can be reduced by a class of methods like dynamic control of system clock speed by operating system scheduler.

So, they come up with some right scheduling algorithms for taking advantage of adjusting the clock speed at a fine grain, which CPU energy can be substantial saved with a limited impact on performance.

B. Reduce the energy consumption of non-critical jobs

In [1], the authors study the slack time for non-critical jobs, extends their execution time and reduces the energy consumption without increasing the tasks execution time as a whole. Moreover, they consider Green Service Level Agreement which is increasing task execution time within an affordable limit. They also develop a heuristic scheduling to reduce energy consumption of a tasks execution, and their main concern is discussing the relationship between energy consumption and task execution time.

C. Case study of DVFS on processor clock frequencies at older and recent platform

In [2], the authors discuss how the clock frequency of a processor is decreased to allow a corresponding reduction in the supply voltage using Dynamic voltage and frequency scaling (DVFS). This power consumption reducing can lead to significant reduction in the energy for a computation, especially for memory-bound workloads. But thats only for older platform; recent developments in processor and memory technology have brought in the saturation of processor clock frequencies. Then, they analysis how reduce processor clock frequency at DVFS on older and recent platforms. The result is that this method actually increases energy usage on most recent platforms; it can only effective on older platforms.

D. VM work placement to cut cost

In [5], the paper study the multi-dimensonal resource usage states of physical machines. It talks about the problem of the natural imbalance of jobs to tend to favor a specific resource, such as CPU or memory. This results in jobs that fills up all phyiscal memory but very little CPU and vice versa. The paper proposes a set covering solutuion to reduce energy usage by using the minimal number of physical machines. This naturally results fitting the incoming jobs in such a way that packs high memory intensive jobs with high CPU intensive jobs. However, this solution is generally efficient and does not apply to workloads where you only have one type (CPU or Memory intensive workloads).

III. Defintions

First, we have the set of jobs, J:

$$J = \{j_1, j_2, j_3, ..., j_i\}$$

We also have the set of computing clusters (DVFS aware):

$$C = \{c_1, c_2, c_3, ..., c_j\}$$

Each DVFS cluster supports 2 energy states, taking from set V:

$$V = \{v_1, v_2\}$$

v_1 is used for non-critical jobs, v_2 is used for critical jobs. The notion of criticality is apparent throughout the relation between each sets. Essentially, noncritical jobs will get placed on noncritical clusters, which runs in noncritical energy state. Critical jobs will be ran on clusters that is of critical energy state v_2. Each cluster is either noncritical, or critical, and runs in the corresponding energy state.

We use a simple formula to calculate energy cost per job:

TFT: Task Finish Time
TST: Task Starting Time

$\Omega = (TFT_{j_i} - TST_{j_i}) * p_{v1}$
$\Omega = (TFT_{j_i} - TST_{j_i}) * p_{v2}$

The first term calculates the total execution time of job j_i in ms, and the second term is $price\ per\ ms$ for voltage state v_1/v_2. TFT and TST are specified by the customer.

Last, we use two decision variables to denote whether a job is critical or noncritical:

$$x_{j_{iv1}} \in \{0, 1\}$$

$$z_{j_{iv2}} \in \{0, 1\}$$

Next, we will introduce our framework and talk about some specific considerations that is nonobvious and nontrivial.

IV. Model

A. Optimization

$$\min \sum_{i=1}^{|J|} ((TFT_{j_i} - TST_{j_i}) * p_{v1} * x_{j_{iv1}}) + ((TFT_{j_i} - TST_{j_i}) * p_{v2} * z_{j_{iv2}}) + (\epsilon_{penalty} * ms_{penalty_{j_i}})$$

The first two terms simply calculate the energy cost of job j_i, while the last term is the penalty costs when job j_i is not finished on time.

B. Constraints

(1) $x_{j_{iv1}} + z_{j_{iv2}} = 1$ for $i = 1, 2, 3, ..., |J|$

Each job is either critical, or noncritical, but not both.

(2) $ms_{penalty_{j_i}} \leq TFT_{j_i} - TST_{j_i}$ for $i = 1, 2, 3, ..., |J|$

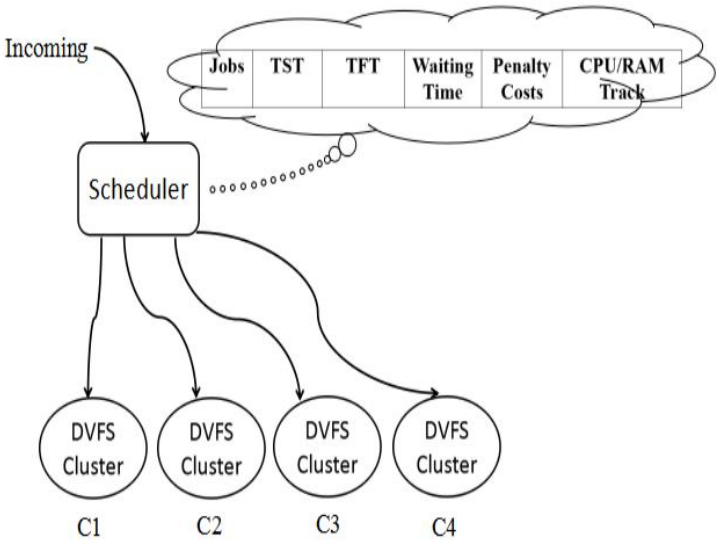
The penalty time should not be more than expected execution time. This also ensures each job gets scheduled and executed. If a job is not scheduled, then $ms_{penalty_{j_i}}$ is infinite.

V. Extended Model

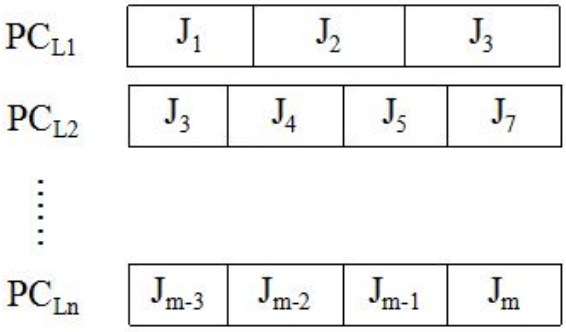
In addition, we include an incoming jobs queue for the set

$$J = \{j_1, j_2, j_3, ..., j_i\}$$

These incoming jobs will first be stored in a centralized queue.



Using greedy algorithm, the scheduler allocate jobs to low energy cluster first. At each step we pick the jobs with the earliest deadline. The general goal that we will try to sqeeze all jobs into low energy cluster and only use high energy cluster when certain $threshold$ are reached. In essense, if it is possible to finish all tasks without resorting to use of more expensive resources, then we will take that path.



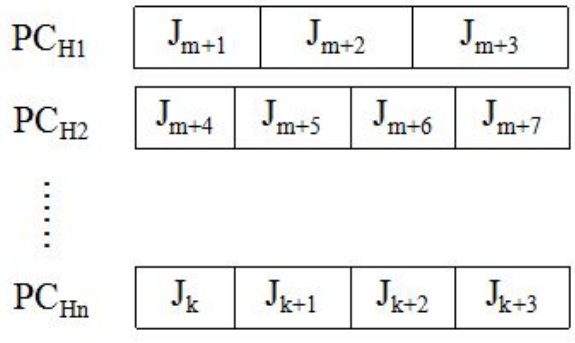
First, we will sort the incoming job queue J in ascending deadline:

$$j_{1deadline} \leq j_{2deadline} \leq j_{3deadline} \leq ... \leq j_{ndeadline}$$

If there are many jobs have the same deadline at our centralized queue. We execute them in their order at our queue (FIFO). The sorting facilitates our greedy scheduling algorithm.

If there are one or more jobs that extend past their respective deadlines and also cross over the $threshold$, then the set of those jobs is transfered to a high energy cluster queue(s). If

there are no available resources to take on those jobs, then that set of jobs will wait in queue.



At the next scheduling period, we will make another determination to see if any high energy clusters are available to take on those jobs. While in queues and during execution, the jobs may accumlate deadline cost penalities. Thus, in the $threshold$ for decision making, as mentioned before, will include two variables. The first variable is the penality cost accured, and the second variable is the data transfer cost.

$$\text{penalty cost} = \text{data transfer cost} + \sum_{i=1}^{|J|} (\epsilon_{penalty} * ms_{penalty_{j_i}})$$

The data transfer cost is simply the cost resulting from the fact that transfering a job from low energy to a high energy cluster results in a time delay. That delay can be thought of as an increase in expected executation time of the job. Generally, if the transfer time is more than the time gained by switching from a low energy cluster to a high energy cluster, then we should reject switching to a high energy cluster. Each job's expected executation is intitally calculated as follows.

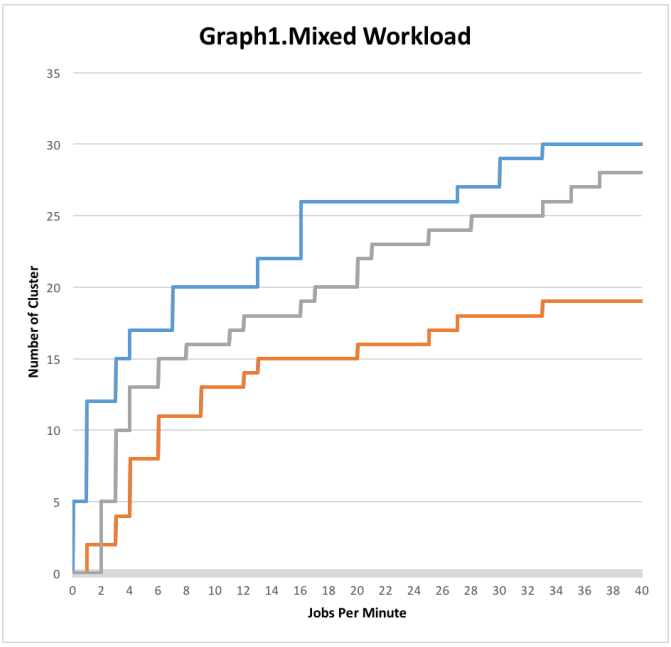
$$j_{i_{execution}} = \frac{CPI * InstructionCount}{ProcessorFrequency}$$

It is calculated twice, once for low energy cluster and once for higher energy cluster.

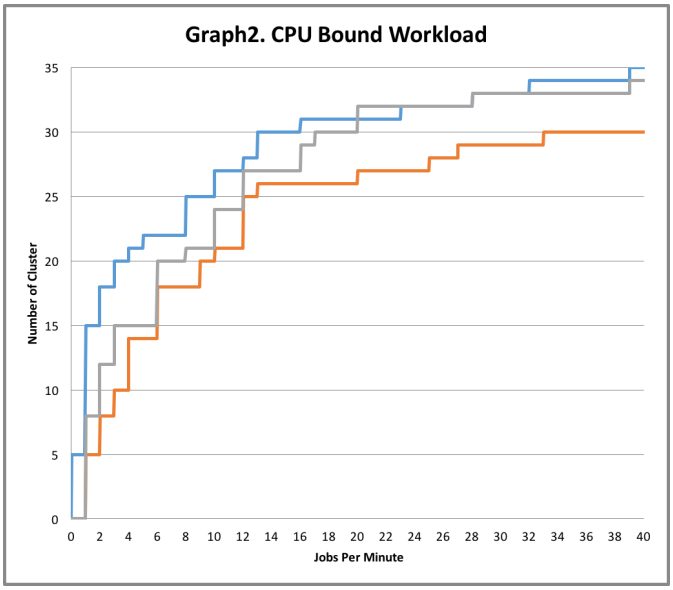
But, there is still an important issue, if a random job $j_{r_{deadline}}$ has a relatively later deadline, and new jobs with earlier deadlines keep coming into our centralized queue, that will cause a big problem, which is keeping this random job j_r in our queue for a long time, thus will have a negative influence at user experience. So, the variable "waiting time" is used to let those jobs we mentioned before have a high priority at our centralized queue, and can be executed first. When the waiting time of a random job j_r is higher or equal to its execution time, we promote a high priority to this job j_r . If there are two jobs have been promoting a high priority and have the same waiting time and execution time which equals "deadline minus enter time", then we execute them in their order at centralized queue.

VI. Results

In our testing, we use Microsoft Azure cloud and rented 35 computers to test three workload types. The first type is a typical workload, which consist of an mixed amount of memory, storage, and CPU intensive jobs. The second type is CPU bound jobs. This job type is also fairly common among cloud computing jobs. The last type is memory intensive jobs, to represent jobs like MapReduce. We compare our scheduler to 2 other typical industry schedulers.

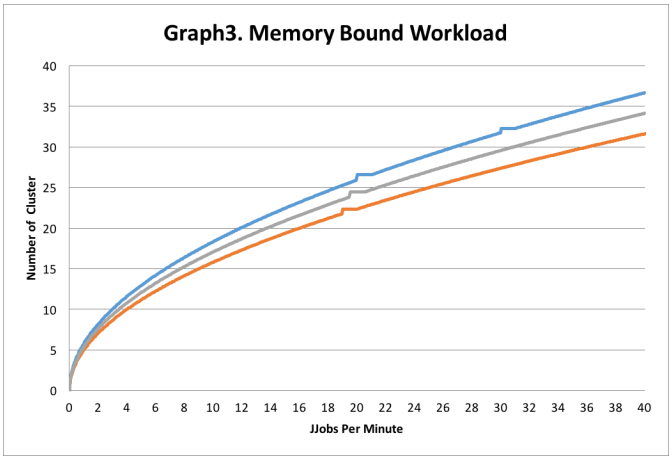


Mixed workload shows our scheduler outperforming the other two scheduler.



Under CPU bound workload conditions, all schedulers

show similar performance.



Again, all schedulers demonstrate similar performance.

VII. Conclusion

Our scheduling method demonstrates moderate improvments when dealing with CPU intensive jobs and mixed jobs workloads. This is reflected in the fact that in a worload of 40 jobs, our scheduling method used the least amount of computing resource. This is already a cost improvement since idle resouces can be used to run more jobs or turned off to further improve power use. In the case of CPU bound job types, our scheduling method showed a small improvement, but this is expected as energy saving is diffculty when a great amount of CPU resource is required for proper execution of a job. In the last type of jobs, all scheduling methods show very similar results. This is also expected. Moreover, our scheduling method did not attempt to minimize memory or network usage.

References

[1] Wang, Lizhe, et al. ”Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS.” Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on. IEEE, 2010.

[2] Le Sueur, Etienne, and Gernot Heiser. ”Dynamic voltage and frequency scaling: The laws of diminishing returns.” Proceedings of the 2010 international conference on Power aware computing and systems. 2010.

[3] Weiser, Mark, Brent Welch, Alan Demers, and Scott Shenker. ”Scheduling for reduced CPU energy.” In Mobile Computing, pp. 449-471. Springer US, 1994.

[4] Ren, Shaolei, Yuxiong He, and Fei Xu. ”Provably-efficient job scheduling for energy and fairness in geographically distributed data centers.” Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on. IEEE, 2012.

[5] [5] Li, Xin, et al. ”Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center.” Mathematical and Computer Modelling 58.5 (2013): 1222-1235.