On the Utility of DVFS for Power-Aware Job Placement in Clusters

Jean-Marc Pierson[1] and Henri Casanova[2]

[1] IRIT, University of Toulouse, Toulouse, France, pierson@irit.fr
[2] Dept. of Information and Computer Sciences, University of Hawai'i at Manoa, Honolulu, Hawai'i, U.S.A., henric@hawaii.edu

Abstract. Placing compute jobs on clustered hosts in a way that optimizes both performance and power consumption has become a crucial issue. Most solutions to the *power-aware job placement problem* boil down to consolidating workload on a small number of hosts so as to reduce power consumption which achieving acceptable performance levels. The question we investigate in this paper is whether the capabilities provided by DVFS, i.e., the ability to configure a host in one of several power consumption modes, leads to improved solutions. We formalize the problem so that a bound on the optimal solution can be computed. We then study how the optimal, if it can be computed, and its bound vary across scenarios in which hosts provide various degrees of DVFS capabilities. We rely on a DVFS model that we instantiate based on real-world experiments. Our approach thus quantifies the potential improvements that hypothetical job placement algorithms can hope to achieve by exploiting DVFS capabilities.

1 Introduction

The problem of efficiently allocating resources among competing jobs and users on clusters has received considerable attention. Job scheduling algorithms have been developed that attempt to optimize job placement with respect to criteria related to job performance, system throughput, and/or fairness. Electrical power, although long been ignored in the job scheduling literature, has become a crucial issue for large-scale clusters. Consequently, in recent years many authors have studied the power-aware job placement problem, seeking theoretical as well as practical solutions [18, 9, 17, 16, 2]. A common approach for reducing the power consumption of a cluster is to place jobs so as to consolidate the workload on a small but sufficient number of cluster nodes, temporarily powering off unused nodes. Workload consolidation is enabled by virtual machine (VM) technology, which provides mechanisms to control and adapt resource shares allocated to VM instances, and to quickly migrate VM instances among cluster nodes. In addition to workload consolidation, a possibility is to configure the hardware via Dynamic Voltage Scaling (DVS), Dynamic Frequency Scaling (DFS), or a combination of the two (DVFS), which are now commonplace in modern processors. In this work, we seek to answer the following question: *What is the value added by DVFS when*

used in addition to workload consolidation when solving the power-aware job placement problem? Some authors have proposed solutions that exploit DVFS capabilities [21, 1, 10, 13]. However, obtained results are difficult to compare and do not provide a conclusive answer to the above question. Instead, we opt for an algorithm-agnostic approach and extend an existing formulation of the power-aware job placement problem proposed in the literature so that it accounts for cluster nodes that have DVFS capabilities. This problem can be formulated as an Mixed Integer Linear Program (MILP), which makes it possible to compute a bound on the optimal solution. Studying how the optimal and its bound vary as the DVFS capabilities of cluster nodes are enhanced provides a sound theoretical basis upon which to quantify the utility of DVFS when used in conjunction with workload consolidation. More specifically, our original contributions are:

- We combine and extend the power-job placement formulations in [4] and [23] and study two versions of the optimization problem: (i) optimize performance given a constraint on power consumption; and (ii) optimize power consumption given a constraint on job performance.
- We instantiate a model that captures the trade-off achieved between power consumption and performance using DVFS, based both on previous work and on experiments on a real-world platform.
- We compute the (bound on the) optimal for the optimization problem for several instantiations of the DVFS model, thereby quantifying the added benefit of increased DVFS capabilities.

Our main result is that using DVFS leads only to marginal improvements and that these improvements require only one additional intermediate power mode between the "powered off" and "powered on" modes.

2 Related Work

Since the power-aware job placement problem is NP-hard, most authors aim at designing heuristics. A prevalent technique consists in designing workload consolidation techniques so as to use a limited number of nodes, possibly relying on job migration. In [18, 22], nodes are powered off when not used, and job placement decisions attempt to power a node back on only when absolutely necessary. Similarly, Hoyer et al. [16] propose statistical allocation planning for resource allocation while maintaining each job over a certain threshold on performance reduction. Others use various techniques to make online resource allocation decisions, leading to consolidation of jobs to a minimal set of nodes [17, 20, 15, 2]. In addition to workload consolidation, it is possible to tune the hardware configuration of cluster nodes. For instance, users can specify that some hardware components can be slowed down or powered off for particular jobs [7]. More generally, the use of DVFS has been proposed in conjunction to workload consolidation techniques [9, 21]. In the context of parallel applications, DVFS has also been proposed as way to exploit and mitigate load imbalance and communication delays for the purpose of power consumption reduction [13, 10]. Note

that, like in this work, several authors have formulated various classes of power-aware job placement problems as linear programs [1, 21, 4]. But, to the best of our knowledge, no work to date has exploited such formulations to quantify the theoretical added benefit of using DVFS-enabled nodes.

3 Power-Aware Job Placement with DVFS

3.1 Problem Statement

We define the problem using [23] and [5] as foundations. We consider a cluster with H nodes, or *hosts*, and N jobs that must be placed and allocated resources on those hosts. Each job must be placed on exactly one host. We consider a static workload: no job enters or leaves the system and job resource needs are constant. Sound static job placement provides a good basis for job placement in the case of (more realistic) dynamic workloads. For instance, a static resource allocation can be recomputed periodically to account for changes in the workload. Alternately, a resource allocation can be recomputed for each job arrival/departure.

As in [23], which does not consider power consumption, we consider that hosts provide resources along an arbitrary number of resource dimensions (e.g., CPU time, RAM space, network bandwidth, disk space). Jobs have resource needs along the resource dimensions provided by hosts. We consider two kinds of resource needs: *rigid* and *fluid*. A rigid need denotes that a resource allocation is required. The job cannot benefit from a larger allocation and cannot operate with a smaller allocation. A fluid need specifies the maximum resource allocation that the job could use if alone on a reference host. The job cannot benefit from a larger allocation, but can operate with a smaller allocation at the cost of reduced performance. All that follows assume a single, reference host. A job could have two rigid needs: it could require 50% of the host's RAM and 20% of the host's disk space. The job could have two fluid needs: it could use up to 40% of the host's network bandwidth and up to 60% of the host's CPU time. In this example, the job cannot use both resources fully, for instance because of interdependence between I/O and computation. While not true in all cases [8], for simplicity we assume that rigid resource needs are completely independent from fluid resource needs (and from each other). Job resource needs can be discovered via benchmarking[24], analytical models [14], or runtime discovery [26, 6].

Our metric for quantifying the performance of a particular job placement and resource allocation is the *scaled yield* [23]. For each fluid resource need the yield is defined as the ratio between the resource fraction allocated and the maximum resource fraction potentially used. For instance, if a job has a fluid CPU need of 60% but is allocated only 42% of the host's CPU, then the yield is $42/60 = 0.7$. Following the same rationale as in [23], we assume that the utilizations of all resources corresponding to the fluid needs of a job are linearly correlated. For the previous example, if the job were to be allocated only 20% of the host's I/O bandwidth (i.e., half of what it could potentially use), then it would use only 30% of the host's CPU (i.e., also half of what it could potentially use). The yield of a job is thus identical for all its fluid needs. We thus simply refer to the yield

of a job, which takes values between 0 and 1. A job can, however, specify a QoS requirement as a minimum acceptable yield value. For the earlier example, the yield could be constrained to be higher than 0.4, which means that the CPU fraction allocated to the job would be at least $0.4 \times 60\% = 24\%$. The scaled yield of a job is then defined as:

$$scaled\ yield = \frac{yield - minimum\ yield}{1 - minimum\ yield}\ . \tag{1}$$

The scaled yield of a job thus takes values between 0 and 1. The objective is to maximize the minimum scaled yield over all jobs to optimize both aggregate performance and fairness among jobs [23]. Note that maximizing the average scaled yield is prone to starvation, as seen in [19] in the context of stretch optimization. However, in [23] a second optimization phase is used to maximize the average scaled yield while maintaining the previously maximized minimum scaled yield. The same approach could be used in this work as well.

As in [5], which studies power-aware job placement but considers only two resource dimensions, we assume that each host consumes power depending on utilization of its resources. Unlike [23, 5], we consider that each host provides DVFS capabilities. More specifically, it is possible to reduce the power consumption of a host by reducing the capacity of one or more of its resources (e.g., lowering the number of CPU cycles per seconds by lowering the clock rate). Unlike [23, 5], we allow for heterogeneous hosts. We quantify power consumption of the system as the sum of the power consumptions of all hosts, accounting for the load imposed on each host.

When faced with two distinct objectives (i.e., maximizing yield and minimizing power consumption) one possibility is to optimize a linear combination of them. This approach, albeit commonplace, is problematic because the coefficient of the linear combination must be chosen by the user, and also because the obtained solution is not guaranteed to be Pareto optimal. Instead, we consider two separate single-objective optimization problems:

1. BoundedPower: Maximize the minimum scaled yield given an upper bound on power consumption (i.e., a power budget).
2. BoundedYield: Minimize the power consumption given a lower bound on the minimum scaled yield (i.e., a performance budget).

3.2 Problem Formulation

In this section we formalize both optimization problems introduced in the previous section. We consider H hosts providing a d-dimensional resource and N jobs that must be placed on these hosts. Each host h can operate in $nv_h \geq 2$ distinct power modes. Each power mode corresponds to a different tradeoff between the resources available from the host and its power consumption. We ignore the overhead of modifying a host's power mode, including powering it on and off. In power mode k, host h provides a capacity f_{hjk} for resource dimension j. Rather than quantifying f_{hjk} with absolute resource-specific units, we use a relative

$$\forall h \qquad \sum_k p_{hk} = 1 \tag{2}$$

$$\forall i \qquad \sum_{h,k} e_{ihk} = 1 \tag{3}$$

$$\forall i, h \qquad e_{ih1} = 0 \tag{4}$$

$$\forall i, h, k \qquad e_{ihk} \leq p_{hk} \tag{5}$$

$$\forall i, h, k \qquad 0 \leq y_{ihk} \leq e_{ihk} \tag{6}$$

$$\forall i, h, k, j \qquad w_{ihjk} = r_{ij}(y_{ihk}(1 - \delta_{ij}) + e_{ihk}\delta_{ij}) \tag{7}$$

$$\forall h, k, j \qquad \sum_i w_{ihjk} \leq p_{hk} f_{hjk} \tag{8}$$

$$\forall i \qquad \sum_{h,k} y_{ihk} \geq \hat{y}_i + Y(1 - \hat{y}_i) \tag{9}$$

$$E = \sum_{h,j,k} Power_{hjk}(\sum_i w_{ihjk}) \tag{10}$$

Fig. 1. Optimization constraints.

measure so as to easily account for heterogeneous hosts and/hosts in different power states. For resource dimension j, we identify the maximum resource fraction provided by any host in any power state in the platform. All f_{hjk} values are taken relative to this maximum and are thus between 0 and 1. For all h and j, f_{hj1} is zero, meaning that the first power mode for each host corresponds to being powered down. Job i's resource need along dimension j is denoted by r_{ij} and is relative to the aforementioned maximum. Furthermore, we define δ_{ij} to be 1 if job i's resource need in resource dimension j is rigid, and 0 if the need is fluid. The minimum yield required by job i, i.e., its Quality Of Service requirement, is denoted by $\hat{y}_i$. Finally, the power consumption of host h in power mode k due to resource dimension j is given by a function $Power_{hjk}(x)$, where x is the total resource usage in this resource dimension due to jobs placed on the host.

To formulate both our job placement problems as constrained optimization problems we define the following variables: (i) Y is a rational variable that quantifies the minimum scaled yield over all jobs; (ii) E is a rational variable that quantifies the power consumption of the platform; (iii) p_{hk} is a binary variable that is 1 if host h is in power mode k, and 0 otherwise; (iv) e_{ihk} is a binary variable that is 1 if job i is placed on host h in power mode k, and 0 otherwise; (v) y_{ihk} is a rational variable which quantifies the yield of job i on host h in power mode k; and (vi) w_{ihjk} is a rational variable which quantifies the resource usage due to job i on host h in power mode k for resource dimension j.

We can now write the set of constraints shown in Figure 1, where $i \in 1, \ldots, N$, $h \in 1, \ldots, H$, $j \in 1, \ldots, d$, and $k \in 1, \ldots, nv_h$. Constraint 2 states that a host operates in a single power mode. Constraint 3 states that a job can only be placed on a single host (which operates in a given power mode). Constraint 4 states that a job cannot be placed on a host that is powered down. Constraint 5 states that a job can only be placed on a host in a particular power mode if this host operates in this power mode. Constraint 6 states that the yield of a job can be non-zero only on the host (and its given power mode) on which the job is placed. Constraint 7 defines the resource usage w_{ihjk}, which is non-zero only

if job i is placed on host h that operates in power mode k. For a rigid resource need ($\delta_{ij} = 1$) the resource usage is equal to the job's resource need, while for a fluid need ($\delta_{ij} = 1$) the resource usage is scaled by the job's yield. Constraint 8 states that the resource usage on a host in a given power mode does not exceed the corresponding resource capacity. Constraint 9 simply states that, for each job, the minimum scaled yield, Y, is lower that the job's scaled yield. Finally, Constraint 10 states that the total power consumption of the platform is equal to the sum of the power consumption on all hosts (in their respective power states) along all resource dimensions. One can now formalize BOUNDEDPOWER as maximizing Y subject to the constraints in Figure 1 with an additional $E \leq E_{max}$ constraint. Similarly, BOUNDEDYIELD consists in minimizing E subject to the constraints in Figure 1 with an additional $Y \geq Y_{min}$ constraint.

In this work we make the simplifying assumption that the power consumption of a host is entirely driven by the power consumption of its CPU, i.e., along a single resource dimension. While it would be straightforward to incorporate other sources of power consumption in our problem formalization, models need to be developed for the corresponding $Power_{hjk}()$ functions. In fact, it could very well be that the power consumption of the different resources are not independent, in which case a single $Power_{hk}$ function that models power consumption given the d resource usages on host h in power mode k. For now, given that such models are not available and that the CPU does account for a large fraction of a host's dynamic power consumption, we assume the $Power_{hjk}()$ always returns 0 for $j \neq 1$. Arbitrarily choosing that the CPU resource dimension corresponds to $j = 1$, we define

$$Power_{h1k}(x) = C_{hk}^{min} + C_{hk}^{prop} \times x \ ,$$

where C_{hk}^{min} is the power consumption of host h in power mode k when idle, and C_{hk}^{prop} denotes the proportional increase in power consumption over C_{hk}^{min} due to a CPU load x (i.e the dynamic power consumption). This load is expressed as fraction of the maximum possible number of CPU cycles executed per time unit by a host, over all hosts in all their power states. This model and assumptions were experimentally shown close to real clusters power consumptions [11, 25]. All constraints in Figure 1 are thus linear, and both optimization problems are thus MILPs. One can use a linear solver to compute exact solutions for small instances. For larger instances, one can relax the binary variables (e_{ihk} and p_{hk}) to take rational values between 0 and 1. The obtained solution is generally not feasible in practice, but provides an optimistic bound on the optimal. Note that for resources other than the CPU, the power consumption model may not be linear, in which case the theoretical approach used in this work would not apply.

4 DVFS/DFS Model

In our problem definition a host h is fully specified by C_{hk}^{min}, C_{hk}^{prop}, and f_{h1k}, for $k = 1, \ldots, nv_h$. In this section we explain how we instantiate these values so as to generate problem instances that are representative of real-world platforms.

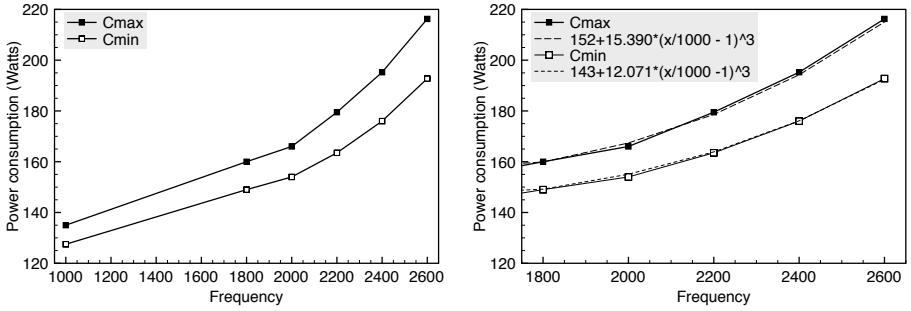

Fig. 2. Left: C^{min} and $C^{max} = C^{min} + C^{prop}$ (in Watts) vs. operating frequency (in MHz). Right: cubic fit on the 1800MHz-2600 frequency range (in MHz).

Several authors have investigated the modeling of power consumption as a function of the frequency-voltage of the processor, which is correlated to compute speed. Following [12], Dynamic Frequency Scaling (DFS), or T-States, allows for running the processor at different frequencies. Dynamic Voltage and Frequency Scaling (DVFS), or P-States, allows for reducing both frequency and voltage, leading to better savings than DFS. Combined DVFS+DFS consists in applying DFS to the lowest power consumption mode available in DVFS.

The most comprehensive processor power consumption study to date is the one in [12]. The authors develop models that relate compute speed to power settings. The derived models are linear for DFS and DVFS, and cubic when both techniques are combined. While these models give power consumption as a function of the frequency, they say nothing regarding the evolution of the minimum and/or maximum power consumption at different frequencies. Consequently, we experimented on a real-world platform (AMD bi-processors dual-core in the Grid5000/Toulouse [3] platform) and we measured values for C^{min} (when hosts are idle) and $C^{max} = C^{min} + C^{prop}$ (when processors run cpuburn) at the 6 available operating frequencies on these hosts so as to generate representative problem instances. Results were consistent across hosts. Figure 2 shows result for one host corresponding to the case when DFS and DVFS are combined, which is what is assumed in all experiments hereafter. The frequency ranges from 1Ghz to 2.6Ghz, and C^{min} and C^{max} range from 127 Watts to 210 Watts, as seen in the graph on left-hand side. As in [12] we generate a cubic model for C^{min} and C^{max} from the experimental data, as seen in the graph on a right-hand side. The model is accurate on the 1.8GHz-2.6GHz range, which is the range used in all experiments hereafter. Using this model, and given a number of power modes and a frequency range, we can thus generate representative problem instances.

5 Numerical Results

5.1 Experimental Methodology

We generate sets of problem instances for BoundedPower and BoundedYield as follows. We generate instances with either $H = 4$ hosts ("small" instances for which we can compute the optimal solution) and $H = 32$ hosts ("large" instances). Given that our goal in this work is solely to investigate the effect of DVFS on job placement, we generate problem instances in which hosts provide only a CPU resource, for which jobs express fluid resource needs. Adding other (rigid) resources would complexify the job placement problem (i.e., in terms of bin packing) but have little impact on the bound on optimal we compute. Each host in the platform can be configured in the same number of power modes (i.e., $nv = nv_h$ does not depend on h). We use $nv \in \{2, 3, 4, 6, 8, 10, 15, 20\}$ for our instances, thus spanning the range from hosts that provide only two power modes (on and off) to hosts that allow for a fine-grain trade-off between compute speed and power consumptions. Although our problem formulation allows for heterogeneous hosts, we found that conclusions regarding the utility of DVFS are identical for homogeneous and heterogeneous platforms. Consequently, we only present results obtained for homogeneous platforms hereafter as the experimental scenarios are much simpler. Rather than picking a number of jobs N, we instead pick a load factor α and generate the smallest number of N random jobs such that the sum of their CPU resource needs is greater than or equal to $\alpha \times H$. For instance, $\alpha = 2$ corresponds to a scenario in which the platform is underprovisioned by (slightly more than) a factor two. For small instances ($H = 4$), we pick α between 1.0 and 1.5, while for large instances ($H = 32$), we pick α between 1.0 and 3.0. Job CPU needs, numbers between 0 and 1, are picked randomly using a truncated Gaussian distribution of mean 0.5. QoS requirements are set to 0 for all jobs.

For each instance specification, i.e., a (H, nv, α) triplet, we generate 100 random host and job specifications. We then generate a BoundedPower instance and a BoundedYield instance. For the BoundedPower instance, we constrain E to be below a fixed value. This fixed value is half of the power consumption obtained when placing jobs on hosts using a simple greedy algorithm (place the next job on the least loaded processor). For the BoundedYield instance, we arbitrarily constrain Y to be above 0.5. For each problem and for all instances we compute the solution of the relaxed MILP, which we denote by LPBound. For small instance, we compute the optimal solution, which we denote by Milp. We use the open-source GLPK linear solver to compute these solutions on an 3.2GHz Intel Xeon processor. We arbitrarily set a timeout value to 10 minutes and declare an instance not solvable if the timeout is exceeded.

5.2 Results for Small Instances

In this section we discuss results obtained on small instances. Our main objective here is to quantify the difference between Milp and LPBound. Figure 3 plots

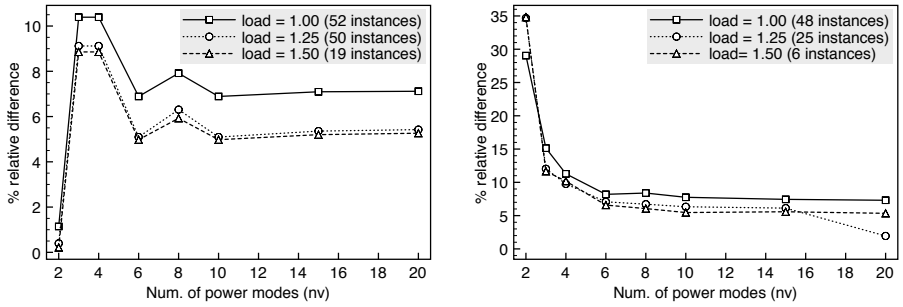

Fig. 3. Percent relative difference between LPBound and Milp vs. nv. Left-hand side: BoundedPower; Right-hand side: BoundedYield.

the average percentage difference between LPBound and Milp computed over successfully solved problem instances as the number of power modes increases, for small BoundedPower instances (left-hand side) and small BoundedYield instances (right-hand side). Each graph contains three curves, one for each value of $\alpha = 1.00, 1.25, 1.50$. The number of successfully solved instances out of the 100 generated instances is indicated in the legend. As expected, as α increases the number of solvable instances decreases since the number of optimization constraints increases. For BoundedPower, we see that LPBound is at most about 11% away from Milp. Interestingly the results for BoundedYield show a different pattern, with the relative different dropping sharply as nv increases. Regardless, in both cases, the relative difference is below 10% for large values of nv. It is these large values that are particularly relevant in this work since they correspond to scenarios in which hosts provide extensive DVFS capabilities. We conclude that LPBound tracks the Milp solution well for small instances. For large instances Milp cannot be computed, but we contend that a better LPBound value (i.e., higher for BoundedPower, lower for BoundedYield) is a strong indication of an opportunity for a better Milp value.

5.3 Results for Large Instances

Table 1 shows results for the averages and standard deviations of the percentage relative improvement of LPBound with $nv = 3$ over the case $nv = 2$ (i.e., no DVFS), both for BoundedPower and BoundedYield. Each row of the table corresponds to a particular α value, and percentage relative improvements are computed between two instances that are completely identical but for the nv values (i.e., they have the exact job sets). The table does not show any results for $nv > 3$ because results are identical to the $nv = 3$ case for all successfully solved instances. Our first important conclusion is thus that configuring hosts in more than 3 power modes does not lead to improved LPBound values.

For the BoundedPower problem, we see that α has no impact on the result. This is because LPBound is computed as the solution to a relaxed MILP,

Table 1. Average relative percentage LPBound improvement for $nv = 3$ instances over $nv = 2$ instances, for the BoundedPower and BoundedYield problems, and percentage of successfully solved instances. Standard deviations are in parentheses.

	BoundedPower		BoundedYield	
α	% imprvmnt	% success	% imprvmnt	% success
1.00	12.14 (8.43e-4)	100	10.83 (1.60e-7)	100
1.25	12.14 (1.05e-3)	100	10.83 (1.45e-7)	100
1.50	12.14 (1.25e-3)	100	9.59 (1.39)	100
1.75	12.14 (1.46e-3)	100	5.02 (1.40)	100
2.00	12.14 (1.53e-3)	100	1.85 (1.11)	88
2.25	12.14 (1.78e-3)	100	0.50 (0.36)	9
2.50	12.14 (2.07e-3)	100	n/a	0
2.75	12.14 (2.07e-3)	100	n/a	0
3.00	12.14 (2.56e-3)	100	n/a	0

meaning that jobs are considered perfectly divisible. In this case, the platform can be conceptually considered as a single host whose compute capacity is simply bounded by the constraint on power consumption E. Changing the α value simply amounts to scaling the yield of all jobs, meaning that the percentage difference between two solutions with different nv values does not change. The situation is different when solving BoundedYield. First, as α increases, GLPK fails to compute solutions. In some cases, these errors are due to 10-minute timeouts, likely due to the fact that the objective function is more complex than for BoundedPower. In other cases, the instances are not solvable, simply because job yields are bounded below by the constraint on the minimum scaled yield. Regardless, a more important observation is that the relative percentage improvement of using $nv = 3$ power modes over using $nv = 2$ power modes decreases as α increases. This is because a larger α implies a higher computational load on the system, which must be accommodated by configuring hosts in higher power modes (while staying within the power consumption budget if possible). Consequently, intermediate power modes are used increasingly less frequently in highly loaded scenarios and DVFS is increasingly less useful.

6 Conclusion

We have extended an existing formulation of the power-aware job placement problem proposed in the literature so that it accounts for cluster nodes that have DVFS capabilities. This formulation is a Mixed-Integer Linear Program, which makes it possible to compute a bound on the optimal solution, and relies on a DVFS model that we have instantiated based on a real-world platform. We have shown on small problem instances that the bound is reasonably close from the optimal solution, thereby indicating the the bound is likely a good indicator of the optimal. For large instances, in which case the optimal cannot be computed in a feasible amount of time, our main results can be summarized as: (i) using $nv > 3$ DVFS power modes never leads to improved LPBound

values; (ii) using $nv > 2$ power modes leads to marginal improvements ($< 15\%$) when solving BoundedPower; (ii) using $nv > 2$ power modes leads to lower improvements ($< 11\%$) when solving BoundedYield, and the improvement decreases quickly as the compute load increases. Assuming that LPBound is a good indicator of the optimal, our overall conclusion is that DVFS is at best marginally useful, and that only one intermediate power mode between "on" and "off" is needed.

References

1. Benoit, A., Renaud Goud, P., Robert, Y.: Sharing resources for performance and energy optimization of concurrent streaming applications, http://hal.archives-ouvertes.fr/hal-00457323/PDF/RR-LIP-2010-05.pdf, RR-LIP-2010-05
2. Berral, J.L., Íñigo Goiri, Nou, R., Julià, F., Guitart, J., Gavaldà, R., Torres, J.: Towards energy-aware scheduling in data centers using machine learning. In: ACM eEnergy. University of Passau, Germany (April 2010)
3. Bolze, R., Cappello, F., Caron, E., Daydé, M.J., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quétier, B., Richard, O., Talbi, E.G., Touche, I.: Grid'5000: A large scale and highly reconfigurable experimental grid testbed. IJHPCA 20(4), 481–494 (2006)
4. Borgetto, D., Casanova, H., Costa, G.D., Pierson, J.M.: Energy-aware service allocation. Tech. Rep. IRIT/RT-2010-7-FR, IRIT (October 2010)
5. Borgetto, D., Da Costa, G., Pierson, J.M., Sayah, A.: Energy-Aware Resource Allocation. In: Proc. of the Energy Efficient Grids, Clouds and Clusters Workshop (E2GC2). IEEE (October 2009)
6. Carrera, D., Steinder, M., Whalley, I., Torres, J., Ayguadé, E.: Utility-based placement of dynamic web applications with fairness goals. In: IEEE Network Operations and Management Symposium. pp. 9–16 (2008)
7. Da Costa, G., Dias De Assuncao, M., Gelas, J.P., Georgiou, Y., LefËvre, L., Orgerie, A.C., Pierson, J.M., Richard, O., Sayah, A.: Multi-Facet Approach to Reduce Energy Consumption in Clouds and Grids: The GREEN-NET Framework. In: ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy), Passau, Germany. pp. 95–104. ACM (2010)
8. Doyle, R.P., Chase, J.S., Asad, O.M., Jin, W., Vahdat, A.M.: Model-based resource provisioning in a web service utility. In: Proc. of the USENIX Symposium on Internet Technologies and Systems (2003)
9. Etinski, M., Corbalan, J., Labarta, J., Valero, M.: Utilization driven power-aware parallel job scheduling. Computer Science - Research and Development 25, 207–216 (2010), 10.1007/s00450-010-0129-x
10. Etinski, M., Corbalan, J., Labarta, J., Valero, M., Veidenbaum, A.: Power-aware load balancing of large scale mpi applications. In: IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing. pp. 1–8. IEEE Computer Society, Washington, DC, USA (2009)
11. Fan, X., Weber, W.D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. In: Proceedings of the 34th annual international symposium on Computer architecture. pp. 13–23. ISCA '07, ACM, New York, NY, USA (2007)
12. Gandhi, A., Harchol-Balter, M., Das, R., Lefurgy, C.: Optimal power allocation in server farms. In: Douceur, J.R., Greenberg, A.G., Bonald, T., Nieh, J. (eds.) SIGMETRICS/Performance. pp. 157–168. ACM (2009)

13. Ge, R., Feng, X., Cameron, K.W.: Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In: SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing. p. 34. IEEE Computer Society, Washington, DC, USA (2005)
14. Gmach, D., Rolia, J., Cherkasova, L., Kemper, A.: Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In: Proc of the 10th IEEE Intnl. Symp. on Workload Characterization. pp. 171–180 (Sept 2007)
15. Hermenier, F., Lorca, X., Menaud, J.M., Muller, G., Lawall, J.: Entropy: a Consolidation Manager for Clusters. Research Report RR-6639, INRIA (2008)
16. Hoyer, M., Schröder, K., Nebel, W.: Statistical static capacity management in virtualized data centers supporting fine grained QoS specification. In: ACM eEnergy. University of Passau, Germany (April 2010)
17. Kamitsos, Y., Andrew, L.L.H., Kim, H., Chiang, M.: Optimal Sleep Patterns for Serving Delay Tolerant Jobs. In: ACM eEnergy. University of Passau, Germany (April 2010), http://netlab.caltech.edu/lachlan/abstract/eEnergySleep.pdf
18. Lawson, B., Smirni, E.: Power-aware resource allocation in high-end systems via online simulation. In: Proceedings of the 19th annual international conference on Supercomputing. pp. 229–238. ICS '05, ACM, New York, NY, USA (2005)
19. Legrand, A., Su, A., Vivien, F.: Minimizing the Stretch when Scheduling Flows of Divisible Requests. Journal of Scheduling 11(5), 381–404 (2008)
20. Niyato, D., Chaisiri, S., Sung, L.B.: Optimal power management for server farm to support green computing. In: CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. pp. 84–91. IEEE Computer Society, Washington, DC, USA (2009)
21. Petrucci, V., Loques, O., Mossé, D.: A Dynamic Optimization Model for Power and Performance Management of Virtualized Clusters. In: ACM eEnergy. University of Passau, Germany (April 2010)
22. Rodero, I., Jamarillo, J., Quiroz, A., Parashar, M., Guim, F., Poole, S.: Energy-efficient application-aware online provisioning for virtualized clouds and data centers. In: first IEEE sponsored International Green Computing Conference (2010)
23. Stillwell, M., Schanzenbach, D., Vivien, F., Casanova, H.: Resource allocation algorithms for virtualized service hosting platforms. Journal of Parallel and Distributed Computing 70(9), 962–974 (2010)
24. Urgaonkar, B., Shenoy, P., Roscoe, T.: Resource Overbooking and Application Profiling in Shared Hosting Platforms. SIGOPS Oper. Syst. Rev. 36(SI), 239–254 (2002)
25. Wang, Z., Tolia, N., Bash, C.: Opportunities and challenges to unify workload, power, and cooling management in data centers. SIGOPS Oper. Syst. Rev. 44, 41–46 (August 2010)
26. Zhu, X., Young, D., Watson, B.J., Wang, Z., Rolia, J., Singhal, S., McKee, B., Hyser, C., Gmach, D., Gardner, R., Christian, T., Cherkasova, L.: 1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center. In: Proceedings of the International Conference on Autonomic Computing (ICAC'08). pp. 172–181 (June 2008)