Optimizing Job Performance Under a Given Power Constraint In HPC Centers

Maja Etinski[†,‡], Julita Corbalan[†,‡], Jesus Labarta[†,‡], and Mateo Valero[†,‡]

[†]Computer Science Department
Barcelona Supercomputing Center
Barcelona, Spain

[‡]Department of Computer Architecture
Technical University of Catalonia
Barcelona, Spain

{maja.etinski,julita.corbalan,jesus.labarta,mateo.valero}@bsc.es

***Abstract*—Never-ending striving for performance has resulted in a tremendous increase in power consumption of HPC centers. Power budgeting has become very important from several reasons such as reliability, operating costs and limited power draw due to the existing infrastructure. In this paper we propose a power budget guided job scheduling policy that maximize overall job performance for a given power budget. We have shown that using DVFS under a power constraint performance can be significantly improved as it allows more jobs to run simultaneously leading to shorter wait times. Aggressiveness of frequency scaling applied to a job depends on instantaneous power consumption and on the job's predicted performance. Our policy has been evaluated for four workload traces from systems in production use with up to 4 008 processors. The results show that our policy achieves up to two times better performance compared to power budgeting without DVFS. Moreover it leads to 23% lower CPU energy consumption on average. Furthermore, we have investigated how much job performance and energy efficiency can be improved under our policy and same power budget by an increase in the number of DVFS enabled processors.**

***Keywords*-power budgeting, parallel job scheduling, DVFS**

I. Introduction

The Top500 list ([1]) of the 500 most powerfull supercomputers updated twice a year reflects a continuous struggle for performance. The current number one ranked system, Jaguar ([2]), comprising of an incredible number of almost 225 thousands cores brings the theoretical peak capability to 2.3 petaflop/s. Striving for performance has resulted in an enormously high peak power draw. Jaguar requires almost 2.8 times the electric power of its predecessor Roadrunner, currently the number two on the list. This difference translates into millions of dollars per year in operating costs. Power estimates for exascale computer power dissipation range from many tens to low hundreds of megawatts ([3]). Hence power consumption is one of the most important design constraints for high performance computing (HPC) centers nowadays. Besides a tremendous increase in cost of ownership, power-awareness in HPC centers is motivated by other reasons such as system reliability and environmental footprint. Starting a year ago the Top500 list is accompanied by the Green500 list ranking the most energy efficient supercomputers in FLOPS/W.

Power budgeting has become very important in the context of power management. A modern HPC center may be faced with a power constraint from several reasons. One group of reasons is motivated by existing infrastructure. For instance, a supercomputing center can have limited power capacity due to the existing power provisioning facilities. Furthermore in a large scale system reliability is a serious issue that is closely related to the cooling facilities. A power budget may be imposed by the existing cooling system. On the other hand power budgeting can be motivated by operating costs. Setting a power constraint guarantees keeping energy consumption, that determines the costs, under a given limit.

Supercomputer theoretical peak capability is used to rank machines on the Top500 list but what matters the most in daily HPC center operation is user satisfaction. In the literature, job performance is usually measured in BSLD (Bounded Slowdown) metric that depends on two components: job's wait time and job's runtime. Job's runtime depends on the machine architecture. On the other hand how long a job submitted to a HPC center will wait for execution is determined by certain factors such as the current load in the center, the scheduling policy, job's requested number of processors, job's requested time, and job's priority. Many efforts have been done to improve job scheduling policies to decrease average wait time.

A job scheduler has a global view of the whole system. It is aware of running jobs and their estimated termination times, and of queued jobs and their potential performance according to different schedules. Since a job scheduler is aware of system activity it can estimate instantaneous power consumption and if necessary apply a power reduction mechanism. CPU power presents one of the major total system power components and it can be easily controlled/reduced in two ways: decreasing the number of running processors and reducing processor performance. Controlling the number of running processors effectively controls CPU power consumption since idle processors can be put in low power modes in which their power consumption is negligible.

However enforcing a power budget in this way leads to great number of jobs blocked in the wait queue and accordingly to an increase in their wait times. Depending on the system size it can happen that the system is underutilized while there are many jobs waiting for execution due to the power constraint.

DVFS (Dynamic Voltage Frequency Scaling) is a widely used technique that trades processor performance for lower power consumption. With DVFS a processor can run at one of the supported frequencies/voltage pairs that are lower than the nominal one. Lower frequency/voltage leads to significantly lower power consumption, hence DVFS presents a useful technique to manage CPU power by running jobs at lower frequencies. As jobs at lower frequencies consumes less, more jobs can execute simultaneously and long wait queues can be avoided. Thus, a job scheduler presents favorable place to integrate power control that keeps power consumption under a given limit while optimizing overall job performance.

In this paper we propose the **PowerBudget- guided** job scheduling policy. Besides parallel job scheduling, it does CPU frequency assignment based on job's *predicted* BSLD. As the goal is to maximize performance for a given power budget DVFS is used only when power dissipation is high enough to endanger the power constraint. Four workloads from systems in production use with up to 4008 processors are simulated to evaluate the proposed approach. The proposed policy shows an improvement in job performance from 20% to 50% over a baseline policy without DVFS.

Furthermore we have investigated job performance and energy efficiency of enlarged systems under our policy for same CPU power budget. Having more DVFS enabled processors introduces a possibility to run more jobs simultaneously within same power budget but at reduced frequencies. A larger system does not necessary dissipate more power since nowadays idle processors can be put in low power modes almost immediately and without an overhead. Furthermore other system components support or should support in near future low power modes suitable for idling ([4]) what will lead to energy proportional computing ([5]). Systems with up to 75% extra processors are simulated to evaluate PowerBudget-guided policy potentials to benefit from additional computational power at lower frequency.

The rest of the paper is organized as follows. The next sections describes the PowerBudget-guided job scheduling. Experimental methodology is discussed in Section 3. Section 4 gives detailed evaluation of the policy proposed in the paper. It is followed by an overview of related work. Finally, Section 6 summarizes the paper.

II. Job Scheduling For a Given Power Budget

We have upgraded the well established EASY backfilling policy ([6]) to support power budgeting. The EASY backfilling is described in the next subsection. It is followed by subsection II-B where we explain our power control policy.

A. The EASY Backfilling

Backfilling-strategies are a set of policies designed to eliminate the fragmentation typical for the FCFS policy. With the FCFS policy a job can not be executed before previously arrived ones although there might be holes in the schedule where it could run without delaying the others. There are many backfilling policies classified by characteristics such as the number of reservations and the prioritization algorithm used in the backfilling queue. The number of reservations determines how many jobs in the head of the wait queue will be allocated such that later arrived jobs can not delay them. When there are less jobs in the wait queue than reservations jobs are executed in FCFS order. If all reservations are used, the algorithm tries to *backfill* jobs from a second queue (the backfilling queue) where jobs are potentially sorted in a different order than by submission time. The EASY-backfilling is one the simplest but still very effective backfilling policy. The EASY backfilling queue is sorted in FCFS order and the number of reservations is set to 1.

MakeJobReservation(J) and ***BackfillJob(J)*** are two major functions in the EASY backfilling implementation. The reservation for the first job in the wait queue is made with ***MakeJobReservation(J)***. If at its arrival time there are enough processors, ***MakeJobReservation(J)*** will start immediately a job. Otherwise it will make a reservation for the job based on submitted user estimates of already running job runtimes. With backfilling policies users are expected to provide runtime estimates in order to allow the scheduler to exploit the unused fragments. It is in user's interest to give an accurate estimate of the runtime as an underestimation leads to killing the job, while an overestimation may result in a long wait time.

The EASY-backfilling is executed each time a job is submitted or when a job finishes making additional resources available for jobs in the wait queue. If there is already a reservation made with ***MakeJobReservation(J)*** , ***BackfillJob(J)*** tries to find an allocation for the job J such that the reservation is not delayed. It means that the job requires no more than the currently free nodes and will terminate by the reservation time or it requires no more than the minimum of the currently free nodes and the nodes that will be free at the reservation time. Jobs scheduled in this way are called *backfilled* jobs.

Traditionally used metric of job performance, BSLD, gives the ratio between the time spent in the system and the job runtime:

$$BSLD = max(\frac{WaitTime + RunTime}{max(Th, RunTime)}, 1) \qquad (1)$$

where $WaitTime$ and $RunTime$ are times that the job spent waiting for the execution and its runtime. Th is a threshold used to avoid impact of very short jobs on the average value. In our experiments the threshold Th is set to 10 minutes as HPC jobs shorter than 10 minutes are classified as very short jobs([7]).

B. PowerBudet-Guided Policy

Since our algorithm assigns a CPU frequency to a job when it is scheduled and it runs at the same frequency during whole execution, we have decided to define our policy as *power conservative.* In a similar way that work conservative scheduling policies manage cpus ([8]), we keep certain amount of power anticipating new arrivals. This concept implies that we start to apply DVFS before a job cannot be started because of the power constraint. On the other hand, when there is no danger of overshooting the power limit DVFS should not be applied in order to maintain execution times achieved at the nominal frequency.

Our policy uses two models when determining job's start time and CPU frequency. A power model is used to estimate power consumption of a job at a frequency/voltage pair (presented in Section III-B). An execution time model gives the new execution times of a job when executed at different frequencies (presented in Section III-C).

The next subsection describes how exactly DVFS aggressiveness is controlled while subsection II-B2 presents the modifications made to ***MakeJobReservation(J)*** and ***BackfillJob(J)*** functions to implement our PB-guided policy.

1) Managing DVFS: Having always in mind that this policy should be integrated in an HPC center, our main aim is to control the performance penalty. Hence, CPU frequency is determined depending on the job's *predicted* BSLD. Predicted BSLD assuming that the job will be run at the frequency f is computed in the following way:

$$PredBSLD = max(\frac{WT + RQ * F(f, \beta)}{max(Th, RQ)}, 1) \qquad (2)$$

where WT is the job wait time according to the current schedule and requested time RQ presents the user runtime estimate. F is a time penalty function that determines how much the execution time is increased due to frequency reduction. Besides the reduced frequency f, the time penalty function F has one more argument β that reflects the job's CPU boundedness.

Table I gives a list of the variables used in the DVFS management policy. A $BSLDth$ threshold is introduced to control DVFS application. Changing the value of this threshold we can control DVFS aggressiveness. Higher $BSLDth$ values allows more aggressive DVFS application that includes use of the lowest available CPU frequencies. Jobs consume less at lower frequencies allowing for more jobs to run simultaneously. Setting $BSLDth$ to a very low value prevents the scheduler from running jobs at reduced frequencies. In order to run at reduced frequency f a job has to satisfies a *BSLD condition* at frequency f. A job satisfies the BSLD condition at frequency f if its predicted BSLD at the same frequency is lower than the current value of $BSLDth$.

Variable	Description
BSLD	Job's Bounded Slowdown due to its wait and execution time
Th	BSLD metric parameter used to avoid impact of very short jobs
WT	Job's wait time (spent in the queue)
RQ	Job's requested time (user runtime estimate)
$F(f, \beta)$	Function that determines impact of frequency scaling on runtime
f	CPU frequency
β	Job's characteristic that determines frequency scaling penalty in time
$PredBSD$	Predicted BSLD based on requested time and CPU frequency
$BSLDth$	Current BSLD target
$P_{current}$	Current CPU power draw (Watts)
P_{lower}	User-specified bound above which frequency scaling is enabled
P_{upper}	User-specified CPU power bound for aggressive frequency scaling
$BSLD_{lower}$	BSLD target when $P_{lower} \leq P_{current} < P_{upper}$
$BSLD_{upper}$	BSLD target when $P_{current} \geq P_{upper}$

Table I
VARIABLES USED WITHIN THE POLICY AND THEIR MEANING

The value of $BSLDth$ is changed dynamically depending on the actual power draw as presented in Figure 1. $BSLDth$ is set based on current power consumption $P_{current}$ that includes power consumed by already running jobs and power that would be consumed by the job that is being scheduled at the given frequency f. P_{lower} and P_{upper} are thresholds that manage *closeness* to the power limit. When CPU power consumption overpasses P_{lower} it means that processors consume a considerable amount of power. When P_{upper} is overshot it is high probability that soon it would not be possible to start a job due to the power constraint. The power thresholds determine the $BSLDth$ threshold: it is set to 0 (for $P_{current} < P_{lower}$), $BSLD_{lower}$ (for $P_{lower} \leq P_{current} < P_{upper}$) or $BSLD_{upper}$ (for $P_{current} \geq P_{upper}$).

Hence, when instantaneous power is not high no frequency scaling will be applied as predicted BSLD according to definition 2 is always higher than 1. When the power consumption starts to increase, $BSLDth$ increases as well leading to frequency scaling. If power draw almost reaches the limit, $BSLDth$ is increased even more to force aggressive frequency reduction using the lowest available frequencies.

2) The EASY Backfilling Extension: As it has been explained before with the EASY backfilling policy a job is scheduled with one of the two functions: ***MakeJobReservation(J)*** and ***BackfillJob(J)***. These functions modified for

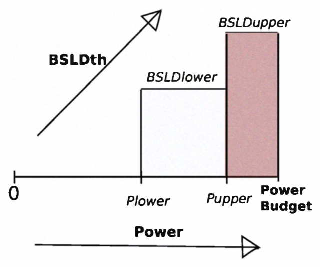


Figure 1. $BSLDth$ as a function of current power

the PB-guided scheduling are shown in Figure 2 and Figure 3 respectively. With the PB-guided scheduling it is not anymore sufficient to find enough free processors to make a job allocation. An allocation has to satisfies the power constraint and the BSLD condition (explained in the previous subsection) if the job will run at reduced frequency (Figure 2 - line 12) or only the power constraint if scheduled for execution at the nominal frequency (Figure 2 - line 20).

```
1:  MakeJobReservation(J)
2:  if alreadyScheduled(J) then
3:    annulateFrequencySettings(J);
4:  end if
5:  $scheduled \leftarrow false$;
6:  $shiftInTime \leftarrow 0$;
7:  $nextFinishJob \leftarrow$
    $next(OrderedRunningQueue)$;
8:  while ($!scheduled$) do
9:    $f \leftarrow F_{lowestReduced}$
10:   while $f < F_{nominal}$ do
11:     Alloc = findAllocation(J,currentTime + shiftInTime,f);
12:     if (satisfiesBSLD(Alloc, J, f) and
        satisfiesPowerLimit(Alloc, J, f) ) then
13:       schedule(J, Alloc);
14:       $scheduled \leftarrow true$;
15:       break;
16:     end if
17:   end while
18:   if ($f == F_{nominal}$) then
19:     Alloc = findAllocation(J,currentTime + shiftInTime,
        $F_{nominal}$)
20:     if (satisfiesPowerLimit(Alloc, J,$F_{nominal}$))
        then
21:       schedule(J, Alloc);
22:       break;
23:     end if
24:   end if
25:   $shiftInTime \leftarrow$
      $FinishTime(nextFinishJob) - currentTime$;
26:   $nextFinishJob \leftarrow next(OrderedRunningQueue)$;
27: end while
```

Figure 2. Making a job reservation

The scheduler iterates starting from the lowest available CPU frequency trying to schedule a job such that the BSLD condition is satisfied at that frequency. If it is not possible to schedule it at the lowest frequency, the scheduler tries with higher ones. Forcing lower frequencies is especially important when there are jobs waiting on execution because of the power constraint (although there are available processors). On the other hand when the load is low, jobs will be prevented from low frequencies by lower $BSLDth$ value. If none of the allocations found in an iteration of the allocation search satisfies all the conditions, then in the next iteration the allocation search looks for an allocation starting from the moment of the next expected job termination (estimated according to requested times).

BackfillJob(J) tries to find an allocation that does not delay the head of the queue and satisfies the power constraint. It also checks the BSLD condition when assigning reduced frequency.

```
1:  BackfillJob(J)
2:  if alreadyScheduled(J) then
3:    annulateFrequencySettings(J);
4:  end if
5:  $f \leftarrow F_{lowest}$
6:  while $f < F_{nominal}$ do
7:    Alloc = TryToFindBackfilledAllocation(J,f);
8:    if (correct(Alloc) and satisfiesBSLD(Alloc, J, f)
      and satisfiesPowerLimit(Alloc,J,f)) then
9:      schedule(J, Alloc);
10:     break;
11:   end if
12: end while
13: if ($f == F_{nominal}$) then
14:   Alloc = TryToFindBackfilledAllocation(J,$F_{nominal}$)
15:   if (correct(Alloc) and
      satisfiesPowerLimit(Alloc, J,$F_{nominal}$)) then
16:     schedule(J, Alloc);
17:   end if
18: end if
```

Figure 3. Backfilling a job

III. Experimental Methodology

A parallel job scheduling simulator, Alvio ([9]), has been upgraded to support DVFS enabled clusters and the power management policy. Alvio is an event driven C++ simulator that supports various backfilling policies. A job scheduling policy interacts with a resource selection policy which determines how job processes are mapped to the processors. In the simulations First Fit is used as the resource selection policy. A description of workloads used in simulations is given in subsection III-A. It is followed by performance and power model explanation.

A. Workloads

Cleaned traces of five logs from Parallel Workload Archive ([10]) are used in the simulations. A cleaned trace does not contain flurries of activity by individual users which may not be representative of normal usage. Table III summarizes workload characteristics. The second column gives the number of processors that the system comprises of. We have simulated 5000 job part of each workload given in the third column of Table III. The parts are selected so that they do not have many jobs removed. The last column of the table gives average workload BSLD when the EASY backfilling without power constraints is used as the job scheduling policy.

The CTC log contains records for IBM SP2 located at the Cornell Theory Center. The log presents a workload with many large jobs but with relatively low degree of parallelism. SDSC and SDSC-Blue logs are from the San Diego Supercomputing Center. The SDSC workload has less sequential jobs than the CTC workload while run time distribution is very similar. In the SDSC-Blue workload there are no sequential jobs, to each jobs is assigned at least 8 processors. The LLNL-Thunder workload contains several months worth of accounting records in 2007 from a system installed at Lawrence Livermore National Lab. Thunder was devoted to running large numbers of smaller to medium size jobs. More information about workloads can be found in Parallel Workload Archive ([10]).

B. Power Model

CPU power consists of dynamic and static power. Dynamic power depends on the CPU switching activity while static power presents various leakage powers of the MOS transistors.

The dynamic component equals to:

$$P_{dynamic} = ACfV^2 \tag{3}$$

where A is the activity factor, C is the total capacity, f is the CPU frequency and V is the supply voltage. In our model we assume that all applications have same average activity factor, i.e. load-balanced applications with a similar β (defined below) across all nodes. Hence, dynamic power is proportional to the product of the frequency and the square of the voltage. According to [11] static power is proportional to the voltage:

$$P_{static} = \alpha V \tag{4}$$

where the parameter α is determined as a function of the static portion in the total CPU power of a processor running at the top frequency. All the parameters are platform dependent and adjustable in configuration files. In our experiments static power makes 25% of the total active CPU power at the highest frequency.

We have used DVFS gear set given in Table II. The last row of the table presents normalized average power dissipated per processor for each frequency/voltage pair.

Our power management is performed at high level. As frequency is assigned to a job statically for whole execution overheads due to transitions between different frequencies are negligible. The same applies for entering or exiting low power modes.

f(GHz)	0.8	1.1	1.4	1.7	2.0	2.3
V(V)	1.0	1.1	1.2	1.3	1.4	1.5
Norm(P)	0.28	0.38	0.49	0.63	0.80	1.0

Table II
DVFS GEAR SET

As nowadays there are low power modes in which power consumption is negligible energy consumed by a workload assumes that idle CPUs dissipate no power. Moreover, a design of a whole system that consume negligible power while idling has been proposed ([4]). Low power modes give an opportunity to explore full potential of system enlarging and frequency scaling.

C. Frequency Scaling Impact On Runtime

In this section we describe a model used in definition of *predicted* BSLD to determine new runtime at reduced frequency. Usually, due to non-CPU activity (memory accesses and communication latency) the increase in time is not proportional to the change in frequency. The β metric, introduced by Hsu and Kremer ([12]) and investigated by Freeh et al. ([13]), compares the application slowdown compared to the CPU slowdown:

$$T(f)/T(f_{max}) = \beta(f_{max}/f - 1) + 1 \tag{5}$$

Different jobs experience different execution time penalty depending on their CPU-boundedness. Theoretically, if an application would be completely CPU bound then its β would be equal to 1. β equals to 0 means that execution time is completely independent of the frequency. The β parameter has values between 0 and 1, although it is generally lower than 0.5. The largest difference that was observed between any two β values for same application but different frequency is 5%. Hence, that the β value of an application is an application characteristic and it does not depend on the amount by the frequency was reduced. β values used in this work are extrapolated from measurement results reported in related work ([13]). While sequential applications from the NAS, SPEC INT and SPEC FP suites have averages of 0.40, 0.59 and 0.71 respectively, parallel benchmarks from the NAS PB suite have a variety of β values from 0.052 of FT class A to 0.466 of SP class C running on 8 nodes. Having in mind that the nodes were connected by very slow 100Mb/s network applications have shown less sensitivity to frequency scaling than they would have with a faster network. Hence, we have assumed β values to be slightly higher. We have generated β for each job according to the following normal distributions:

- if the number of processors is less or equal to 4: $N(0.5, 0.01)$

- if the number of processors is higher than 4 and less or equal to 32: $N(0.4, 0.01)$
- if the number of processors is higher than 32: $N(0.3, 0.0064)$.

We have investigated two cases:

- β is known in advance: at the moment of scheduling real β is used (Section IV-A1)
- β is not known in advance: at the moment of scheduling the worst case is assumed ($\beta = 1$) i.e. requested time is scaled by $\beta = 1$ and runtime is scaled by real beta (Section IV-A2).

Since it would be difficult in practice to know β of each job in advance, it is important to see how conservative approach of assuming β of each job to be 1 affects the performance. The worst case of 1 is assumed in order not to have jobs killed because of overpassing their requested times.

IV. RESULTS

First we have evaluated our policy for original system sizes that correspond to the workload logs described in Section III-A. As a baseline we have assumed a policy that enforces same power budget without DVFS. With the baseline policy jobs are scheduled with the EASY backfilling with an additional power constraint that prevents a job from being started if it would violate the power constraint. Besides the oracle case when β values are known at the moment of scheduling, we have examined case when beta values are not known in advance. Finally, system dimensioning has been explored.

The last three columns of Table III have been obtained for scheduling with the EASY backfilling without any power limitation. Taking into account average system utilizations without a power constraint (**Utilization**) we have decided to set power budgets to 80% of the maximum CPU power of the corresponding system. Maximum CPU power is consumed when all system processors are busy running at the nominal frequency. Percentage of time that a workload spends above the power budget with the EASY backfilling without power constraint is given in Table III (**Over PB**). Imposing 80% power budget decreases job performance tremendously. The power constraint severely penalizes average job wait time. For instance, average wait time of the CTC workload without power budgeting is 7 107 seconds while with the baseline power budgeting it becomes 26 630 seconds. The LLNLThunder average wait time originally was 0 seconds and in the baseline case it has been increased to 7 037 seconds. Hence BSLD job performance degradation is very high.

The policy parameters P_{lower} and P_{upper} are set to 60% and 90% of the workload power budget respectively. After some initial tests we have decided to use the average BSLD of the workload without power constraints (avg(BSLD)) for the parameter $BSLD_{lower}$. In this way the $BSLD_{lower}$ value is set to a workload dependent value. The parameter $BSLD_{upper}$ is two times higher, it is set to $2 * avg(BSLD)$. We have simulated all workloads without power budgeting to get the original average BSLD values. These values without a power constraint are given in the last column of Table III (**Avg.BSLD**). The policy parameters are same for all reported results.

A. Original System Size

In this section we evaluate the PB-guided policy applied to original size systems. In the first subsection β values are used at the moment of scheduling when estimating frequency scaling penalty on user's runtime estimate. Accordingly β values are reflected in BSLD prediction and in job scheduling via their impact on updated user runtime estimates. The second subsection investigates the impact of not-knowing β in advance.

1) Evaluation of PB-guided Policy: The average BSLD per workload for the baseline and our power control policy are given in Figure 4(a) (lower values are better). Although application of DVFS increases job runtime, having more jobs executing at the same time reduces wait time. Average wait times of baseline and PB-guided policies are given in Figure 4(b). Our policy under the power constraint improves significantly overall job performance for all workloads. In the case of the LLNLThunder workload performance is almost twice better.

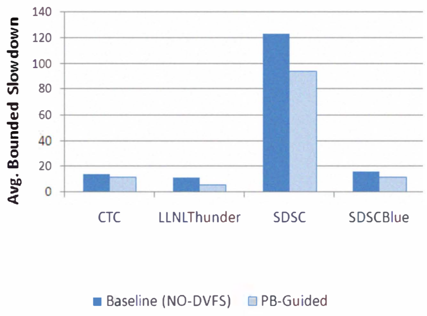
Figure 4(c) shows CPU energies consumed per workload with the two polices. The values are normalized with respect to the case when all jobs are run at the nominal frequency (the highest one). The energy consumed with the PB-guided policy is significantly reduced as a result of DVFS use in the PB-guided policy. Baseline energy is equal to 1 since it assumes that all jobs are executed at the nominal frequency.

Figure 5 gives system utilization and normalized instantaneous power of the baseline (the upper graphics) and the PB-guided (the lower graphics) policies for the LLNLThunder workload over its execution. Instantaneous power is normalized with respect to the power budget. It can be remarked that the PB-guided policy has slightly lower instantaneous power and the workload execution takes shorter time. Furthermore system utilization is higher (in Figure 5(b) - it is always lower than 80%). With the PB-guided policy utilization reaches 100% running jobs at reduced frequencies.

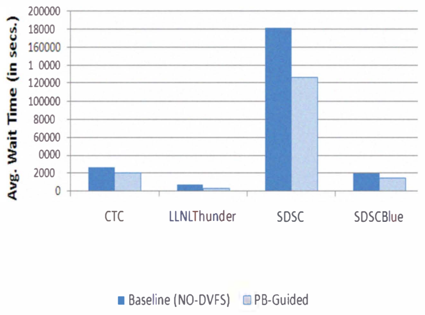
2) Impact Of Unknown Beta: Interestingly, we have observed that assuming more conservative β better performance is achieved. Scaling requested time with $\beta = 1$ and runtime with a lower value introduces more inaccuracy. It has been remarked that inaccurate estimates can yield better performance than accurate ones ([14]). By multiplying real estimates by the factor $\beta * (f_{max}/f - 1) + 1$ jobs with long runtimes can have large runtime overestimation at schedule time leaving at runtime larger 'holes' for backfilling shorter

Workload	Number of CPUs	Jobs	Utilization	Over PB	Avg.BSLD
CTC	430	20 - 25	70%	72%	4.66
LLNLThunder	4008	20 - 25	80%	89%	1.00
SDSC	128	40 - 45	85%	95%	24.91
SDSCBlue	1152	20 - 25	69%	74%	5.15

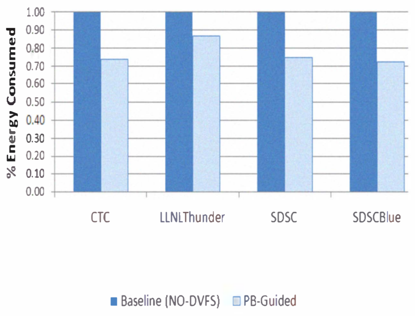
Table III
WORKLOADS



(a) Performance - Avg.BSLD

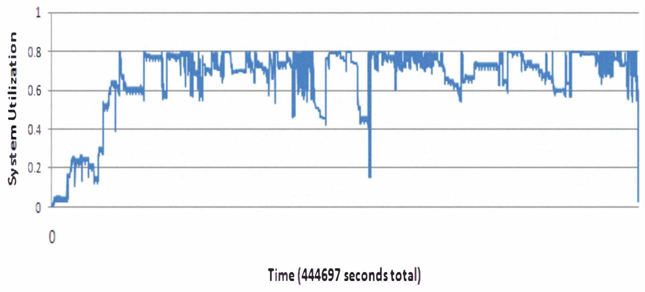


(b) Performance - Avg.WT

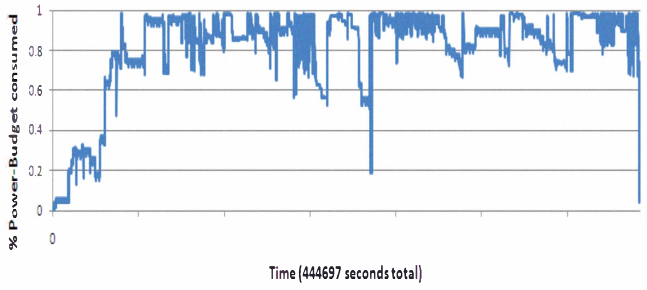


(c) Normalized Energy

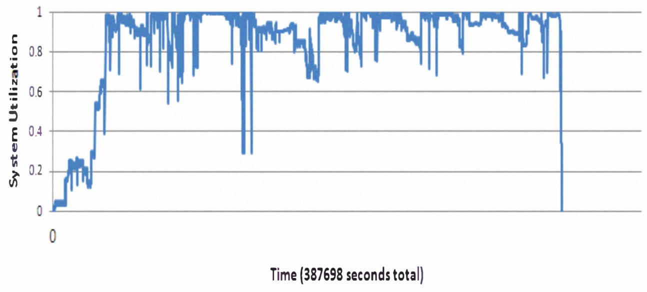
Figure 4. Original System Size



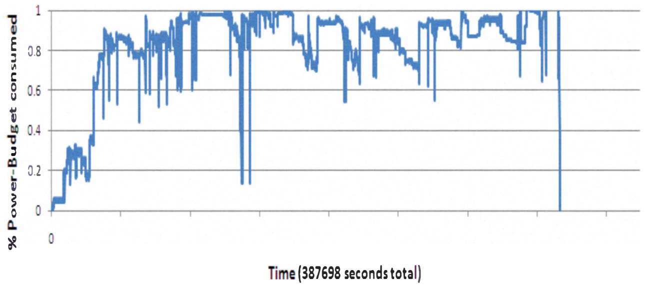
(a) Baseline - System Utilization



(b) Baseline - Instantaneous Power



(c) PB-Guided - System Utilization



(d) PB-Guided - Instantaneous Power

Figure 5. Comparison of baseline and PB - guided policies

jobs. As a result average slowdown and wait time may be lower.

In Table IV results are given for the case when β values are not known in advance (no β) and for the oracle case when it is assumed that they are known in advance (β). Given values are normalized with respect to corresponding baseline case values. All workloads expect SDSC-Blue achieve better performance (**Normalized Avg.BSLD**) when β values are not known in advance. This can be explained by an increase in the number of backfilled jobs that has been observed (**Number of Backfilled Jobs**).

When β is not known, the most conservative case of $\beta = 1$ is assumed and accordingly assigned frequencies are higher on average (see column **Avg.Freq**). As higher frequencies have lower penalty on runtimes, it presents the second reason for better performance with unknown β.

Workload	Normalized Avg.BSLD		Normalized Avg.WT		Avg.Freq		Normalized Energy		Number of Backfilled Jobs	
	no β	β	**no β**	β	no β	β	no β	β	no β	β
CTC	0.80	0.79	**0.63**	0.75	1.5	1.4	0.740	0.738	3924	3923
LLNLThunder	0.33	0.51	**0.28**	0.47	2.07	1.9	0.884	0.865	3830	3651
SDSC	0.62	0.76	**0.53**	0.70	1.6	1.5	0.755	0.744	4103	3944
SDSC-Blue	0.86	0.75	**0.73**	0.76	1.5	1.4	0.727	0.724	3611	3550

Table IV
COMPARISON OF UNKNOWN AND KNOWN β FOR ORIGINAL SYSTEM SIZE

Regarding energy consumption for the two cases presented in column **Normalized Energy** savings are slightly higher when β values are known at the moment of scheduling. This is explained again by lower frequency selection in the oracle case as jobs running at lower frequencies consume less CPU energy.

B. System Oversizing

In this section we have tested our policy for 20%, 50% and 75% enlarged systems. Workloads and power budgets have stayed the same while system sizes have been increased. In this way we have investigated what is the effect of having more processors under same circumstances as before.

Performance for various system sizes is shown in Figure 6(a). It improves very fast with the system size increase. For instance, for 20% increase in system size, CTC's average BSLD drops from 11.23 to 6.38. For 75% system size increase, its average BSLD with the PB-guided policy is 2.15 in the oracle case and 1.79 when β values are not known in advance.

When applied to a larger system, the PB-guided policy assigns lower frequencies as it can be seen in Figure 7(a) that shows average processor frequency per workload depending on the system size. In a larger system within the same power budget wait times are shorter (Figure 7(b)). Accordingly the predicted BSLD values are lower allowing more aggressive frequency reduction. Although lower frequencies have higher negative impact on execution times, the decrease in wait times leads to significantly better BSLD values. Moreover executing jobs at lower frequencies results in energy consumption reduction that is reflected in operating costs. Energy consumption for enlarged systems normalized with respect to the energy consumed when all jobs are run at the nominal frequency is given in Figure 6(b). For 20% system size increase energy reduction of more than 30% is achieved for all workloads except LLNLThunder (it achieves energy savings of 20%).

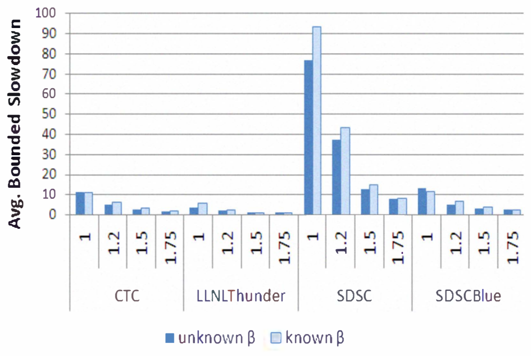
Behavior of our policy for increased system sizes when β values are not known in advance is same as for the original system size. Due to more conservative β value used at the scheduling time assigned frequencies are higher than in oracle case (Figure 7(a)). Performance is slightly better and energy consumption is slightly higher when β values are not known in advance (Figure 6).

V. RELATED WORK

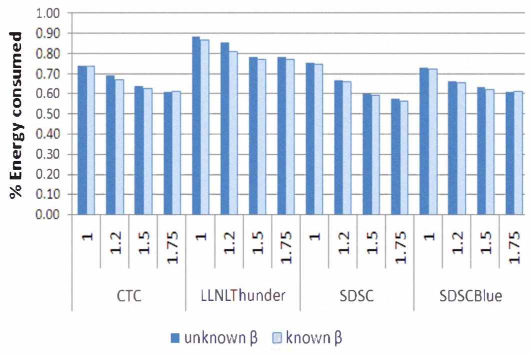
Power budgeting has been examined in the following works. Isci et al. have investigated chip level power management that maintains a chip level power below a specified power budget ([15]). Several different policies that assume per-core DVFS have been proposed and their impact on performance has been evaluated by simulations. Policies that allocate dynamically power budget of an application between processor and memory have been proposed ([16]). Wang et al have explored an approach to shift power among servers based on their utilization while controlling the total cluster power to be lower than a constraint. ([17]). Frequency assignment is performed at very fine grain and it is driven by the model predictive control theory. Applying this approach to a large scale system would involve very high overhead. Lefurgy et al have been presented a technique for high density servers that uses feedback control to keep the system within a fixed power constrained ([18]). This work, same as the previous one, limits only processor performance to control whole-server power consumption.

In HPC there are several groups of works that deal with power consumption. The first group presents works at the application level. Power profiling of parallel applications under different conditions has been done ([19], [20]). Although they have only reported power and execution time on specific platforms for various frequencies or numbers of nodes, they have given a valuable insight into relations between CPU frequency, power and execution time. Power reduction systems based on previous application profiling have been proposed ([21], [22], [23]). Several runtime systems that apply DVFS in order to reduce application energy consumption have been implemented ([24], [25], [26]). These systems are designed to exploit certain application characteristics such as load imbalance of MPI applications or communication-intensive intervals. Therefore, they lead to power reduction only when applied to certain jobs. Nevertheless, these runtime systems are complementary to our approach as far as power allocated to a job at the scheduling time is not violated. This implies that the highest frequency available to them might be lower than the nominal one in some cases.

The second group targets system level power management of large scale systems. Lawson et al have tried to decrease supercomputing center power dissipation by powering down

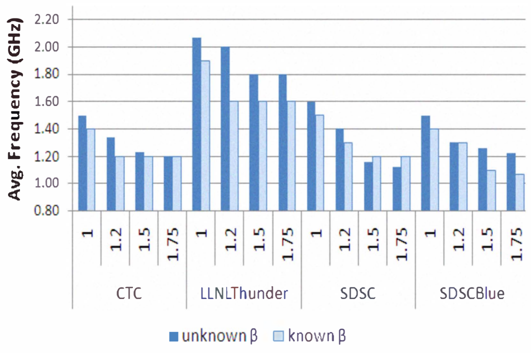


(a) Performance - Avg.BSLD

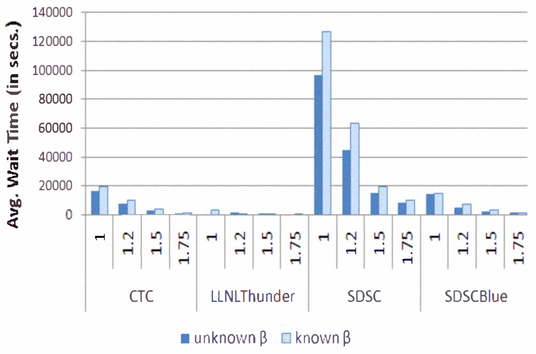


(b) Normalized Energy

Figure 6. Performance and energy with PB-guided policy for various system sizes



(a) CPU frequency distribution



(b) Average wait time

Figure 7. Various system sizes for two β scenarios

some nodes according to two proposed policies ([27]). The EASY backfilling has been used as the job scheduling policy. This approach has affected BSLD seriously in cases of high load. An empirical study on powering down some of system nodes has been done ([28]). A resource selection policy used to assign processors to a job has been designed to pack jobs as densely as possible and accordingly to allow for powering down unused nodes. Kim et al have proposed a power aware scheduling algorithm for bag-of-tasks applications with deadline constraints on DVFS enabled clusters ([29]). It gives a frequency scaling algorithm for a specific type of job scheduling with deadline constraints that is not common in HPC centers. Fan et al have explored the aggregate power usage characteristics of large collection of servers ([30]). The authors have also investigated possibility of energy saving using DVFS that is triggered based on CPU utilization. Elnozahy et al. have proposed policies for server clusters that adjust the number of nodes online as well as their operating frequencies according to the load intensity ([31]). Pinheiro et al have also decreased power consumption by turning down cluster nodes under low load ([32]). Since shutting a node or bringing it back up has taken non-negligible amount of time, it has not been recommendable to simply shut down all unused nodes.

VI. Conclusions

In this paper it has been shown how performance in HPC centers under a power constraint can be improved significantly using DVFS. Not only that careful DVFS application gives better performance but it leads to energy reduction decreasing operating costs. Our PB-guided job scheduling policy uses lower CPU frequencies allowing more jobs to run simultaneously. Depending on instantaneous power and additional adjustable thresholds, the policy assigns frequency to each job at the scheduling time. In the best case it achieves almost two times better performance than the baseline when applied to the original size systems. As in practice it would be difficult to know in advance the impact of frequency scaling on execution time of each job, we have investigated the most conservative case when β is assumed to be 1 at the scheduling time. The results show that it can lead to even better overall performance.

Furthermore, power budgeting with our policy increasing system size has been evaluated. The ideas has been to test performance of more processors at lower frequencies under same power constraint. Increasing system size by only 20% while applying the proposed PB-guided policy leads to significant improvement in performance and to up to 35% savings in CPU energy. This has shown the importance of proper DVFS system dimensioning.

Future work will include a detailed evaluation of the policy parameters. Various values for P_{lower} and P_{upper} thresholds that determine when the policy should start to use/use aggressively DVFS will be tested. Also, the impact of $BSLD_{lower}$ and $BSLD_{upper}$ parameters that govern frequency selection will be investigated more deeply.

Acknowledgment

We would like to thank Barry Rountree for comments that helped improve this paper. This work has been supported by the Spanish Ministry of Science and Education under contract TIN200760625C020), by the IBM/BSC MareIncognito project under the grant BES-2005-7919 and by the Generalitat de Catalunya (2009-SGR-980)

References

[1] http://www.top500.org/, "The top500 list."

[2] http://www.nccs.gov/computing resources/jaguar/, "The jaguar supercomputer."

[3] P. Henning and A. B. W. Jr., "Trailblazing with roadrunner," *Computing in Science and Engineering*, vol. 11, pp. 91–95, 2009.

[4] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," *SIGPLAN Not.*, vol. 44, no. 3, pp. 205–216, 2009.

[5] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[6] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.

[7] R. Kettimuthu, V. Subramani, S. Srinivasan, T. Gopalsamy, D. K. Panda, and P. Sadayappan, "Selective preemption strategies for parallel job scheduling," *Int. J. High Perform. Comput. Netw.*, vol. 3, no. 2/3, pp. 122–152, 2005.

[8] R. Smirni, E. Rosti, E. Smirni, G. Serazzi, and L. W. Dowdy, "Analysis of non-work-conserving processor partitioning policies," in *In IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 1995, pp. 165–181.

[9] F. Guim and J. Corbalan, "A job self-scheduling policy for hpc infrastructures," in *JSSPPS '08: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 2008, pp. 51–75.

[10] http://www.cs.huji.ac.il/labs/parallel/workload/, "Parallel workload archieve."

[11] J. Butts and G. Sohi, "A static power model for architects," *Microarchitecture, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*, pp. 191–201, 2000.

[12] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction," in *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*. New York, NY, USA: ACM, 2003, pp. 38–48.

[13] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835–848, 2007.

[14] D. Tsafrir and D. G. Feitelson, "The dynamics of backfilling: solving the mystery of why increased inaccuracy may help," in *IEEE International Symposium on Workload Characterization (IISWC)*, San Jose, California, October 2006, pp. 131–141.

[15] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 347–358.

[16] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A performance-conserving approach for reducing peak power consumption in server systems," in *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*. New York, NY, USA: ACM, 2005, pp. 293–302.

[17] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *14th International Conference on High-Performance Computer Architecture (HPCA-14 2008), 16-20 February 2008, Salt Lake City, UT, USA*. IEEE Computer Society, 2008.

[18] C. Lefurgy, X. Wang, and M. Ware, "Power capping: a prelude to power shifting," *Journal Cluster Computing*, pp. 183–195, April 2008.

[19] V. Freeh, F. Pan, N. Kappiah, D. Lowenthal, and R. Springer, "Exploring the energy-time tradeoff in mpi programs on a power-scalable cluster," *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 4a–4a, April 2005.

[20] S. Kamil, J. Shalf, and E. Strohmaier, "Power efficiency in high performance computing," *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, April 2008.

[21] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale mpi programs," in *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 1–9.

[22] V. W. Freeh and D. K. Lowenthal, "Using multiple energy gears in mpi programs on a power-scalable cluster," in *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. New York, NY, USA: ACM, 2005, pp. 164–173.

[23] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, "Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster," *ipdps*, vol. 0, p. 340, 2006.

[24] C. hsing Hsu and W. chun Feng, "A power-aware run-time system for high-performance computing," *sc*, vol. 0, p. 1, 2005.

[25] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," *sc*, vol. 0, p. 14, 2006.

[26] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, "Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs," *sc*, vol. 0, p. 33, 2005.

[27] B. Lawson and E. Smirni, "Power-aware resource allocation in high-end systems via online simulation," in *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*. New York, NY, USA: ACM, 2005, pp. 229–238.

[28] J. Hikita, A. Hirano, and H. Nakashima, "Saving 200kw and 200 k dollars per year by power-aware job and machine scheduling," *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, April 2008.

[29] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters," *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pp. 541–548, May 2007.

[30] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2007, pp. 13–23.

[31] E. N. (mootaz Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *In Proceedings of the 2nd Workshop on Power-Aware Computing Systems*, 2002, pp. 179–196.

[32] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *In Workshop on Compilers and Operating Systems for Low Power*, 2001.