EAR: Energy management framework for supercomputers

Julita Corbalan
Barcelona Supercomputing Center (BSC)
Politechnic University of Catalonia (UPC)
Barcelona, Spain
julita.corbalan@bsc.es

Luigi Brochard
Marseille, France
luigi.brochard@gmail.com

***Abstract*—EAR is an energy management framework which offers three main services: Energy accounting, energy control, and energy optimisation. It is composed by five components: EAR library (EARL), EAR submission plugin, EAR daemon (EARD), EAR database manager (EARDBD), and EAR Global Manager (EARGM). This paper overviews main EAR features as energy management framework but focuses on EARL energy optimisation and the coordination with EARGM to provide energy control. EARL is a dynamic, transparent, and lightweight runtime library that provides energy optimisation and control: It can be configured to boost the frequency of energy efficient applications (`MIN_TIME_TO_SOLUTION` policy) or to save energy by reducing the frequency up to a maximum performance degradation (`MIN_ENERGY_TO_SOLUTION` policy). EARL optimises energy by selecting the *optimal* CPU frequency, based on the energy policy selected and application runtime characteristics without any application modification or user input. To do that, EARL relies on two main technologies: the standard MPI profiling interface (PMPI)+LD PRELOAD to offer a transparent execution, and a new algorithm, called DynAIS, which allows EARL to dynamically (and transparently) detect the iterative regions (outer loops) of MPI applications. DynAIS detects repetitive regions at runtime. Once the iterative pattern is detected, EARL computes a set of metrics per iteration, *application signature*, and, together with the *system signature*, applies time and power models to estimate the execution time and power with the list of available frequencies. System signature is a set of coefficients per-node computed during a learning phase executed at EAR installation. Given time and power projections, EARL selects the best frequency based on policy settings. To be highly scalable, EARL uses a totally distributed strategy where each node works totally independent, minimising the interference in application execution. Evaluation has been done in a real production system running applications up to 1024 cores. Results show how EARL adapts frequency to runtime application characteristics and energy policy. EARL has reported speedups of up to 6% when running cpu intensive jobs with `MIN_TIME_TO_SOLUTION` policy and energy savings up to 8% when running more memory intensive jobs with `MIN_ENERGY_TO_SOLUTION` policy. (including EARL overhead). EARGM is a cluster wide component continuously monitoring the energy and power consumed in the cluster. EARGM configuration specifies which conditions have to be considered as warnings, with different levels of criticality, and actions to be taken to adapt the energy or the power to the specified one. Actions could be relative/soft-actions such as be more aggressive with energy policy settings, or hard actions such as reducing the maximum frequency in all the nodes. Since EARL is aware of application characteristics, decisions are taken locally according to warning levels and application running in nodes avoiding the necessity of centralised decisions and its decisions will hurt as little as possible the performance of the system.**

***Index Terms*—Energy, runtime , frequency selection, DVFS, HPC applications, MPI, MPI+OpenMP**

I. Introduction

Power and energy management in data centers is an important research topic. Different solutions to improve power usage effectiveness (PUE) have been proposed at different contexts: from cluster level to architectural solutions. At the software layer, DVFS has been the low level technology mainly used to reduce the power (or energy) consumed by applications. However, frequency selection is a hard work since it not only depends on the application and the architecture, it depends on runtime factors such as the input data, the specific set of nodes, the number of cores, etc. Fig. 1 shows the execution time and power of some of the applications used in this paper. We can clearly observe how the impact of frequency variation depends on the application but also it is not the same on time and power. We can also observe how it is not linear, making it difficult to determine the impact of energy.

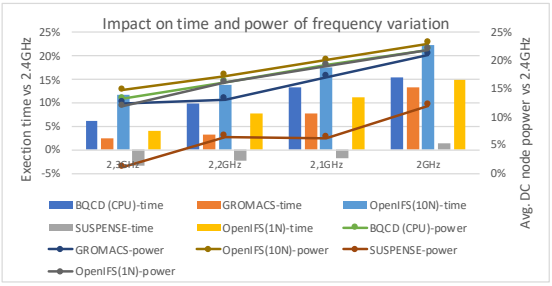

Fig. 1. Impact of frequency variation on execution time (bars) and power (lines) (F_{ref}) vs (F_i)

Apart from frequency selection, dealing with energy and power consumption from the point of view of the system is something more complex that needs to be considered as another resource. In this paper we present EAR, an energy management framework for HPC with a set of components

This work has been done in the context of the BSC-Lenovo collaboration agreement and and partially supported by the Spanish Ministry of Science and Technology through TIN2015-65316-P project, by the Generalitat de Catalunya (contract 2017-SGR-1414)

and a nature similar to a resource manager. EAR offers energy optimization, accounting, and control of a limited resource: the system energy. As far as we know, EAR is the first energy management framework including these three services for HPC environments working coordinated and without significant dependencies with external components. This paper presents EAR framework but focuses on two main components: The EAR library (EARL) and the EAR Global Manager (EARGM). Apart from these two components, EAR includes a submission plugin (SLURM spank plugin is provided in the current version), EAR node daemon (EARD), and EAR data base manager (EARDBD). This work already presents DynAIS, an algorithm included in EARL for outer loops detection on the fly without application modification, the main strengthness against other runtime solutions. EARL includes two energy policies `MIN_TIME_TO_SOLUTION`, targeted to reduce execution time of energy efficient applications by boosting their frequency, and `MIN_ENERGY_TO_SOLUTION`, targeted to save energy by reducing frequency.

Main EAR contributions are: First, DynAIS has demonstrated the ability to dynamically identify application internal structure with very lilttle overhead for real applications with millions of MPI calls. Second, the integration of DynAIS with EARL offers a powerful runtime system for energy optimisation allowing EARL to be self-evaluated. Third, EARL works totally distributed, having no limitations or bottlenecks that would prevent the scalability of jobs with EARL. Fourth, EARL+EARDs+EARGM offers a powerful combination offering energy optimisation and control. Again, with the distributed EAR approach, there is no centralized data and no complex and no scalable, algorithms. Our approach is based on detecting situations that must be fixed and just broadcast the warning level to nodes and let them to automatically react and redirect the situation locally.And last, but not least, as energy management framework, EAR provides a complete solution for energy management.

The evaluation demonstrates EAR introduces a very little overhead (or no overhead in some cases) and shows how the two energy policies included with EAR produces execution time reductions up to 13% when using the `MIN_TIME_TO_SOLUTION` policy and energy savings up to 8% with `MIN_ENERGY_TO_SOLUTION` policy. The rest of the paper is organised as follows: Section II describes EAR as energy management framework. Section III presents EAR library in detail. Section IV describes EAR Global Manager, the component in charge of dynamically adapting global settings to fit in power& energy capping policies. SectionV evaluates EARL in terms of performance and energy savings. Section VI presents the related work. And finally, section VII presents conclusions and future work.

II. EAR OVERVIEW

EAR offers three main services: Energy accounting, energy/power control, and energy optimisation. Energy accounting and control apply independently of the type of job. Energy optimisation is offered by EARL and only enabled for MPI or hybrid MP+OpenMP applications. These three features are offered by five components: EAR library (EARL), EAR submission plugin, EAR daemon (EARD), EAR database manager (EARDBD), and EAR Global Manager (EARGM).

- EAR Daemon is a per-node daemon executed with privileges and running in all the compute nodes. It provides three services: (1) It provides EARL access to privileged metrics and low level functionality such as changing the CPU frequency. (2) It receives energy control requests from EARGM and, either notifies EARL when there is a running job with EARL in the node, or it applies the required actions when running a non-EARL job. (3) It also offers energy accounting for all the jobs (adding EARL metrics in case the job is executing with EARL) and periodic node power monitoring.
- EARL is the component implementing energy optimisation policies. It is deeply described in section III
- EAR submission plugin (SLURM Spank plugin [3]). A key issue to be a real suitable approach in production systems is to be *easy to use*. EAR includes a SLURM Spank plugin to make the EAR deployment and job submission in a system with the SLURM Workload manager transparent to users and simple to sysadmins. The SLURM Spank plugin is a very powerful mechanism that allows extending sbatch/salloc/srun options. EAR framework can be configured to load EARL by default with all the applications or not. When EARL is enabled by default, jobs are automatically loaded and configured with EARL by doing standard sbatch/salloc/srun submission. When EAR is disabled by default, loading it is as simple as adding `--ear=on` to the slurm command. Moreover, we have included options to modify energy policy (`--ear-policy=policy`), energy policy settings (`--ear-policy-th=value`), etc. A complete user guide can be found in [18].
- EAR Database manager. Energy accounting is an important topic for HPC centers and is deeply related to optimisation and control: we need to monitor before understand and evaluate. EARDBD provides three services: (1) buffering, (2) aggregation of data, and (3) replication for reliability. EARDBD has been designed for large HPC centers (and optional for small systems) taking into account the amount of connections and volume of data to be reported in large HPC centers. Many instances of EARDBDs can be executed in the system, each one taking care of a set of nodes.The EAR DB is implemented as a MySQL DB, in particular we use MariaDB [31]
- EAR Global Manager monitors the energy consumed in the system and apply a global energy/power capping policy to control the energy/power consumed. A single instance is executed in the system and it works coordinated with EARL and EARDs. The detection of energy/power warning situations to be addressed is done globally, but specific actions are taken locally by EARL instances (or with EARDs in case the job running in node is not loaded

with EARL). EARGM is deeply described in section IV
EAR is highly configurable and it supports adding components incrementally. Typical configurations are, for instance, focusing on energy optimisation, in this case EARL+EARDs+EAR plugin must be installed. If energy accounting is desired, the EAR DB must be created and EARDs must be installed together with EAR plugin, however, EARL is not required in this case. Depending on the system size, EADBDs will be also required. EAR is configured through an ear.conf configuration file following a very similar approach to slurm.conf, making easy the installation for sysadmins used to SLURM. EAR also includes several commands and tools to report information stored in the DB by job[.step] (`eacct`), by node(`ereport`), etc, as well as an energy control command (`econtrol`) to manually change global energy settings. This command is mainly targeted to systems where EARGM is manually configured. EAR is an open source product and it is going to be distributed under the LGPL license. EAR software and documentation will be available at [17].

III. EAR LIBRARY

EARL is a dynamic, transparent, and lightweight runtime library that provides energy optimisation and control: It can be configured to boost energy efficient applications or to save energy by reducing the frequency up to a maximum performance degradation (controlled by EAR). EAR selects the *optimal* CPU frequency, based on the energy policy selected, without any application modification or user input. Fig. 2 shows the main internal EARL phases: (1)MPI interception, (2) Loop detection through DynAIS, (3) Application Signature computation, (4) Time and power models projections, and (5) energy policy execution (using these projections). EARL

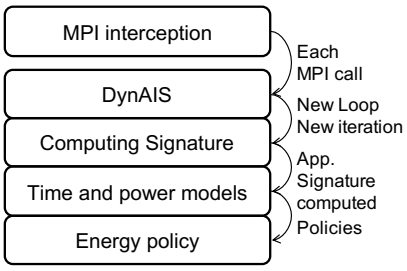

Fig. 2. EAR software stack

relies on two main technologies: the standard MPI profiling in interface (PMPI)+LD_PRELOAD to offer a transparent loading and execution of the library, and a new algorithm, called DynAIS, which dynamically (and transparently) detects the iterative regions (outer loops) of MPI applications (or hubrid MPI+OpenMP). EARL generates a runtime sequence of events based on MPI calls (together with its arguments) which is the input for DynAIS. DynAIS compares each event with the last N[1] events and returns one *state*. DynAIS states are interpreted by EARL to identify loops. Once the iterative pattern has been detected, EARL computes a set of metrics per iteration, *application signature*, that uniquely identify the application dynamic behaviour (which depends on specific nodes, input data, configuration, etc) and applies time and power models to estimate the execution time and power with the list of available frequencies. Given time and power projections, EARL selects the *best* frequency given the policy settings. Since EARL is aware of the application structure, it self-evaluates the frequency selected, and revert it (or re-apply the policy) if needed. Moreover, EARL detects any change in the application, because running a new loop or because it changes its behaviour, applying again the energy policy with new context and metrics.

To be highly scalable, EARL uses a totally distributed solution where each node works totally independent, minimising the interference in application execution and avoiding any additional network traffic and/or additional synchronisation. Evaluation has demonstrated application metrics are quite similar in all the cases and variations in frequency selected are mostly motivated by differences in node characteristics and never more than 1 pstate. As part of the future work we are introducing the mechanisms to coordinate application tasks and evaluating the impact on performance of per-node selection vs per-job selection.

A. DynAIS: ***Dyn****amic* ***A****pplication* ***I****terative* ***S****tructure detection algorithm*

One of the main contributions of EALR with respect other proposals is the ability of EARL to transparently detect the iterative structure of applications on the fly. Given no user intervention is required nor source code modifications, energy optimisations and control provided by EARL can be applied to any already existing binary, from the first job in the system, no matter whether it has been executed before or not, and there is no possibility that users manipulates applications, or system criterias, on its own benefit, which sometimes happens when using historic information. Many scientific applications are composed by nested parallel loops. EARL is able to detect this structure with only one requirement: at least one MPI call must be placed inside the iterative code. EARL converts each dynamic MPI call in a event identifier by combining representative values for each MPI call: the call type, the source addresses and the destination addresses . These events become the input for the Dynamic Application Iterative Structure algorithm (DynAIS). Fig. 3 shows how DynAIS is integrated with EARL: Each MPI call is catch by EARL through the PMPI interface. For each call (except init and finalize functions) the internal EAR_mpi_call is called with the selection of arguments relevant for each MPI call (null values are used for those functions with less number of parameters than required). These values are combined to created a single dynamic event as DynAIS input.

DynAIS doesnt introduce any additional semantic to what these values are, the semantic of what a new loop or new iteration means is something the upper level tool using DynAIS has to decide. In the case of EARL, events are similar

[1]N is part if the DynAIS configuration and it is called the Dynais window size

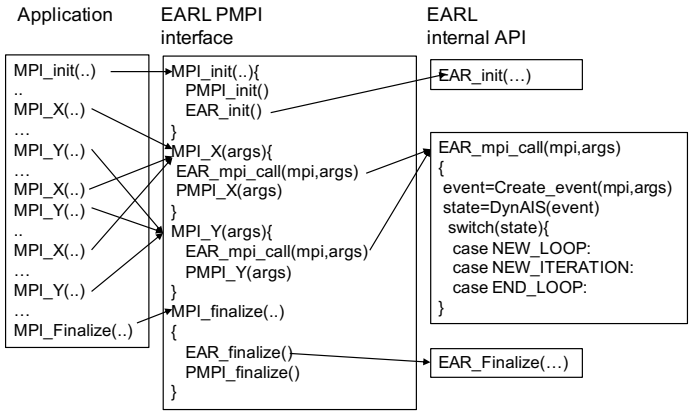

Fig. 3. DynAIS integration in EARL

to breakpoints in the code, and with the help of DynAIS they are used to detect repetitive sequences of dynamic MPI calls. DynAIS detects when a new event belongs or not to an already detected repetitive sequence of values (a loop) and returns the according status associated to this event: `NEW_LOOP` detected, `NEW_ITERATION` (to an already detected loop), and `END_LOOP`. DynAIS is implemented as a multi-level algorithm : A basic DynAIS algorithm detects repetitive sequences of discrete events and it is self-invoked with new arguments when a new loop is detected. Multi-level DynAIS is able to detect nested repetitive sequences (loops with loops inside) up to M[2] levels. In this paper, when we refer to DynAIS, it means Multi-Level DynAIS. Basic DynAIS uses concepts that comes from the context of signal processing [5] where sequences of analogyc values where compared to detect patterns in waves. We have adapted these ideas to the specific case of using discrete values, where two values are equal only when the difference is 0. DynAIS stores a maximum of N events in circular buffers and compares each new event with previous ones, computing the differences. A first version of this algorithm was presented by one of the authors of this paper in [6]. Current DynAIS algorithm includes the utilisation of AVX512 instructions to speedup it as much as possible. However, the capacity of basic DynAIS algorithm to identify loops depends on N, and at the same time the overhead introduced is proportional to it. For real applications, with millions of MPI calls and loops with more than 20.000 events per iteration[3], the buffer size needed to identify outer loops was prohibitive because of the overhead introduced. Because of that, we designed a multi-level DynAIS version with two goals: maintain the same precision than basic DynAIS and minimise as much as possible the overhead.

Fig. 4 shows the main steps for the multi-level DynAIS. The multi level version extends the DynAIS input from one event to pairs of two values. Two *events* are equal when the two values of the pair are equal. The idea behind this decision is to identify not only repetitive sequences of events but also repetitive sequences of *loops*, where a loop is now defined not only by the first event in the loop but also with its length, and the level at which it is . To be able to use the same basic algorithm independently of the level, each level uses its own data structures, and each *event* coming from the application are extended to the pair (*event*,1). With this extension, all the MPI events go to level 0, and only events reporting a `NEW_LOOP` state in level 0 reach level 1. From level 1 to level 2 only `NEW_LOOP` cases are passed and so on. This design implies a high level of abstraction, being able to detect more patterns than with a single big basic DynAIS. With the multi-level DynAIS together with all the optimisations introduced in basic DynAIS, the cost of processing one mpi call was reduced by a factor greater than 1K. Even though it slightly depends on the particular trace file, we have evaluated the average cost per mpi call in 0,1 usecs.

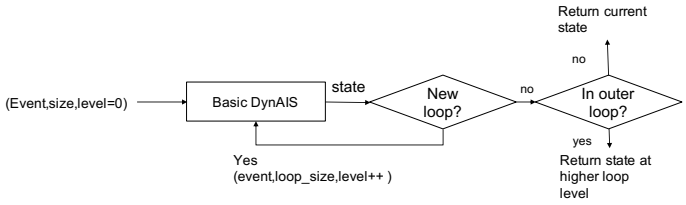

Fig. 4. Outer loop detection with DynAIS

B. EAR modes and EAR states

EARL starts being DynAIS driven but it can switch to periodic mode if DynAIS algorithm is not capable to identify a loop after some reconfigurable time [4]. Since EARL is targeted to be driven by DynAIS, we follow a simple approach when running in periodic mode. In this case, EARL uses a default configuration computing the application signature and applying the energy policy every C number of MPI calls (specified by sysadmin and common to all the applications). We refer to the default, with DynAIS, behaviour as *DynAIS mode* and this case as *Periodic mode*. When running in DynAIS mode, EARL follows the state diagram shown in Fig. 5.

- `FIRST_ITERATION` state. Each time DynAIS reports a `NEW_LOOP` state for a given event, EARL resets internal data structures and measures the execution time of 1 iteration. Based on the iteration time, EARL computes `MIN_ITER`, the minimum number of iterations needed to compute the DC node power. DC node power is measured using DC energy facilities existing in the system. EARL supports the Intel Node Manager [2] power capability for 1s accuracy and a specific EAR development for high accuracy library (100 Hz) based on a specific circuit Lenovo developed for on the new generation of Lenovo water cooled server, the ThinkSystem SD650 [4]. EARL dynamically detects the underlying hardware and computes `MIN_ITER` based on it and the iteration time.

[2]M is part if the DynAIS configuration and it is called the Dynais levels

[3]BQCD loop size is 28.000 events per iteration

[4]A loop with enough granularity to compute the application signature

- EVALUATE SIGNATURE state. After MIN_ITER iterations, EARL computes the application signature and applies the energy policy selected, including frequency changes in case frequency selection is different from the default one. Current signature is saved and compared with the the new one computed after MIN_ITER additional iterations.
- VALIDATE POLICY state. Given both signatures (new one and last one) refer to the same part of the code, EARL detects on the fly a policy failure because of inaccuracies in time and/or power projections or because of applications changes. Anyway, EARL will decide to re-apply the policy or, if changes are quite significant, consider the new application phase as a new loop and go to the FIRST_ITERATION state. If policy decisions are classified as ok, EARL will continue in this state but MIN_ITER will be automatically increased by a factor of 2 to reduce EARL interferences.

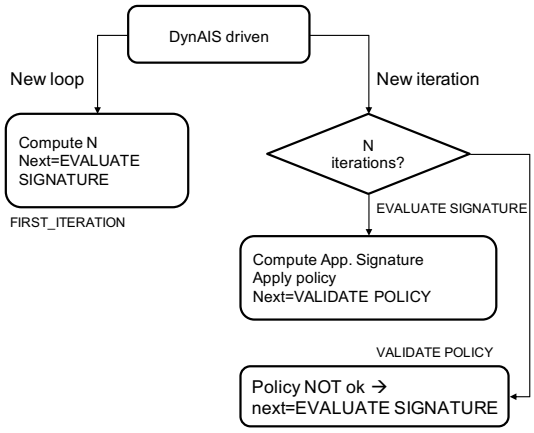

Fig. 5. EAR states when running in DynAIS driven mode

C. Time and power models

When EARL is in EVALUATE SIGNATURE or VALIDATE POLICY states, it computes four metrics describing application dynamic behaviour: Iteration time (seconds), average DC node power, transactions per instructions, and cycles per instructions. These metrics are named *Application signature* and, together with the *System signature*, the are used as the input for the time and power models. These equations were proposed in [29] and [30] and evaluated in [8]. The fact that EARL uses DC node power based on DC node energy and not only CPU energy is one of the differences between EARL metrics and others research works only considering CPU energy. EARL uses PAPI [32] to read cycles and instructions and its own implementation of some IPMI raw commands through the freeIPMI library [33] to measure DC node energy.

System signature is computed per node during EAR installation. It is a per-node matrix of coefficients (A..F) where each point in the matrix is a set of six coefficients used to project time and power from $Freq_i$ to $Freq_j$. At EARL installation, a learning phase is executed. During the training period a set of pre-selected kernels are executed with different frequencies. Kernels are pre-selected to guarantee different applications characteristics are considered from cpu intensive applications to memory intensive. Applications selected are: BT-MZ.C, SP-MZ.C, LU-MZ.C , EP.D, LU.C [9], and UA [10] from the NASPB benchmarks , DGEMM from [11], and STREAM [12]. We run several instances for each case and a linear regression is computed to calculate A,B,C,D,E, and F to project from $Freq_i$ to $Freq_j$ for each i and j in the range of frequencies.

$$Power(f) = A * Power(f_{\text{ref}}) * B * TPI(f_{\text{ref}}) + C \quad (1)$$

$$CPI(f) = D * CPI(f_{\text{ref}}) + E * TPI(f_{\text{ref}}) + F \quad (2)$$

$$Time(f) = Time(f_{\text{ref}}) * (CPI(f))/(CPI(f_{\text{ref}})) * f_{\text{ref}}/f_{\text{n}} \quad (3)$$

D. EARL energy policies

EARL offers two energy policies, MIN_ENERGY_TO_SOLUTION and MIN_TIME_TO_SOLUTION , and a second mode of execution, implemented like a third policy, which is only application monitoring (MONITORING_ONLY "policy"). When selecting this latter "policy", EARL will report application accounting information but the frequency will remain static unless the EARGM dynamically changes node settings (for instance reducing the maximum p_pstate)

E. MIN_ENERGY_TO_SOLUTION

MIN_ENERGY_TO_SOLUTION policy minimizes the energy consumed with a defined limit to the performance degradation. The performance degradation limit is set by the sysadmin in the policy_threshold [5]. The MIN_ENERGY_TO_SOLUTION policy selects the optimal frequency that minimizes energy enforcing (perf_degradation <= policy_threshold) (4). When executed with MIN_ENERGY_TO_SOLUTION policy, applications starts at nominal frequency.

$$perf\_degradation = (T - Tdefault)/Tdefault \quad (4)$$

F. MIN_TIME_TO_SOLUTION

MIN_TIME_TO_SOLUTION reduces the execution time while guaranteeing a minimum ratio between performance benefit and frequency increment that justifies this energy consumption. The policy uses a minimum efficiency set by the sysadmin (policy_threshold option) . For example, if policy_threshold=0.75, EAR will prevent scaling to upper frequencies if the ratio between performance gain and frequency gain do not improve at least 75% (PerfGain >= FreqGain* policy_threshold) (5)(6).

$$PerfGain = (Time - Timenew)/Time \quad (5)$$

[5]Depending on EAR configuration, it can be also set by the user at job submission time

$$FreqGain = (Freqnew - Freq)/Freq \quad (6)$$

When executed with `MIN_TIME_TO_SOLUTION` policy, applications start at a default predefined frequency lower than nominal. For example, given a system with a nominal frequency of 2.3GHz with a default frequency set to 3 p_states less, an application executed with `MIN_TIME_TO_SOLUTION` would start with frequency F_i=2.0Ghz (3 p_states less than nominal). When application signature is computed, EARL computes performance projection for F_{i+1} together with PerfGain(5) and FreqGain(6). If PerfGain is greater or equal than `FreqGain*policy_threshold`, the policy will go on with the next performance projection F_{i+2}. Otherwise, the policy will select the last frequency where the performance gain was enough, preventing the waste of energy.

This policy is the one we recommend when installing EAR in systems requesting high levels of energy efficiency, since it is the most conservative. In this context, it is key to select the *optimal* frequency as default frequency as well as the `policy_threshold`. Default frequency should be selected such the performance of memory bound applications is not affected when running at default frequency. Given that nodes can present differences in power consumption, EAR supports nodes configured with different `policy_threshold` values. For instance, a node consuming more power can have a higher `policy_threshold` than a node consuming less power, adapting locally EARL selections not only to application characteristics but also to node characteristics.

Modifying the value of the `policy_threshold` is one of the key mechanism EAR offers to control the energy. The `policy_threshold` can be dynamically modified by EARGM, forcing the running job to adapt its frequency to the new settings and forcing not enough efficient executions to reduce the CPU frequency. By increasing or reducing the `policy_threshold` EARL will be more or less strict in terms of frequency increase.

IV. EAR Global Manager

EARGM is a cluster wide component continuously monitoring the energy and power consumed in the cluster and taking actions to provide energy/power capping. Energy capping means a maximum energy is allowed for a given period of time. Power capping means a maximum average power is supported. In both cases, EARGM receives the limit and the time *T1* at which this limit is evaluated. In the case of energy capping, EARGM also receives a second long period *T2* indicating the time at which the energy limit applies to. For instance, configuring `T1=1day` and `T2=14 days`, EARGM will compute the energy consumed in the last 14 days and compare this value with the limit, and this value will be updated every day. EARGM can be configured to work in *manual mode* or in *automatic mode*. When configured in manual mode, EARGM will inform the system administrator about energy/power warnings, assuming the sysadmin will manually take the corresponding actions. When configured in automatic mode, EARGM will take the actions to adapt the node settings to the power/energy consumed. Fig. 6 shows the interaction between EARGM-EARD-EARL to provide energy/power control of the cluster. EARGM supports three potential levels of warnings, each one corresponding with an action. In the current version, number of warning levels and actions are statically defined. For manual mode, EARGM sends a mail independently of the warning level including the description of the context. For automatic mode, EARGM will execute the following actions for the different warning levels:

- Warning level 1: In this case, the most soft actions are taken. A message is sent to all EARDs to increase the `policy_threshold`. This is a relative action to the current `policy_threshold` of the node, it is not setting the `policy_threshold` to a fixed value. Since messages is received by EARDs, EAR controls all the nodes. In the case the node is running an job with EARL, the new settings are communicated to the EARL and it automatically adapts the frequency to the new situation. If the application is running a job without EARL, the action is ignored since it is not classified as a critical case. This action will not guarantee an energy/power reduction since it could potentially happen (and in fact it is the target of any system) that all the running jobs are working at the optimum frequency.
- Warning level 2: In this case, a message is sent to all EARDs to both increase `policy_threshold` and reduce the maximum frequency allowed . In this case, given it is the second level of warning, EARDs will automatically apply the action in case the job running in the node is not running with EARL. In case it is a job running with EARL, the new settings are notified and actions are applied by EARL.
- Warning level 3: In this case, since we are very closed to a critical situation, same actions are taken than in warning level 2 but with a higher values. Increasing, again, the `policy_threshold` and reducing again the maximun CPU frequency. Specific values for `policy_threshold` increment or CPU frequency reduction must be set when compiling EARGM. As part of the immediate future work these values will be part of the EARGM dynamic configuration.

After setting a warning level, EARGM waits for a configurable number of `T1` periods to re-evaluate the effectiveness of its actions. After this grace time, it classifies again the warning level, taking into account the last evaluation and applies again the corresponding actions.

V. Evaluation

We have evaluated 5 real applications and 2 benchmarks with the following configurations: without EARL (at 2.4GHz and 2.1GHz), and with EARL+(`MONITORING_ONLY`, `MIN_TIME_TO_SOLUTION` , `MIN_ENERGY_TO_SOLUTION`). Real applications evaluation demonstrate how EARL manages real applications with a significant number of nodes and cores with millions of

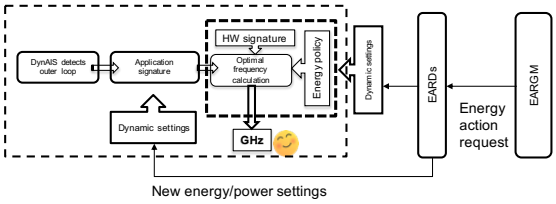

Fig. 6. EARGM coordination with EARDs and EARL

MPI calls. Moreover, one of the applications is an interesting use case since it is the only case (from the evaluated ones) where DynAIS has not been able to detect a pattern and the periodic mode has been used. The two kernels we have included are BT-MZ.C and LU-MZ.D from the NAS-PB [9]. Table I shows specific application configuration in terms of number of nodes, tasks, and cores used in the paper. Two of the applications show different behaviour when changing the input data (BQCD) or same input but different number of cores (OpenIFS). Given these different behaviours, we present a total of 9 use cases. Results show how EARL is able to identify the dynamic use case and adapt the frequency to the specific case as well as the evaluation of the overhead introduced, which is not significant. The two kernels has been selected just to include a basic *cpu use case*, with the BT-MZ kernel, and a *memory use case* with the LU-MZ kernel. BT-MZ.C and LU-MZ.D has been manually extended in the number of iterations to increase the execution time.

- GROningen MAchine for Chemical Simulations (GROMACS) [20] is a molecular dynamics package mainly designed for simulations of proteins, lipids and nucleic acids .
- Berlin quantum chromodynamics program (BQCD) [1] is a Hybrid Monte-Carlo program for simulating lattice QCD with dynamical Wilson fermions.
- SUSPENSE [7] is an improved method for computing incompressible viscous flow around suspended rigid particles using a fixed and uniform computational grid.
- DUMSES [19] is a 3D MPIOpenMP & MPI/OpenACC Eulerian second-order Godunov (magneto)hydrodynamic simulation code in cartesian, spherical and cylindrical coordinates .
- ECMWF OpenIFS [21] is a scientific outreach programme that provides an portable version of the IFS in use at ECMWF for operational weather forecasting.

Applications have been executed in a cluster of Lenovo ThinkSystem SD530 nodes where each node includes two Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz (20c) per node (Hyper-threading is activated but we are not using it), 40 cores in total and 12*8GB dual rank DIMMs per node. For all the experiments, three runs have been executed and we are using the average of the three runs. For a fair comparison, all the executions for each application have been done using the same set of nodes.

Fig. 7 shows the main application metrics used by EARL to select the CPU frequency : the the memory bandwidth

TABLE I
APPLICATIONS

Application	Num.Nodes	MPIs	OpenMPs	Total cores
GROMACS	16	640	1	640
SUSPENSE	25	1000	1	1000
BQCD(CPU)	26	256	4	1024
BQCD(MEM)	26	256	4	1024
OpenIFS(1N)	1	40	1	40
OpenIFS(10N)	10	40	4	400
DUMSES	26	1024	1	1024
BT-MZ.C	4	40	4	160
LU-MZ.D	4	8	20	160

GBs (blue bars) and the CPI (red lines). Legend in the left side shows the GBs (per node) and legend in the right side corresponds with CPI. These metrics have been collected when executing with `MONITORING_ONLY` at 2.4GHz. The graph shows us how there are 4 cases more cpu intensive (GROMACS, BQCD(CPU), OpenIFS(10N), and BT-MZ.C) and four memory cases (SUSPENSE, DUMSES, BQCD(MEM),LU-MZ.D).OpenIFS(1N) is a mixed case since it presents a low CPI and a high memory bandwidth at the same time. Fig. 8 shows the execution time (blue bars) and the average DC node power (red lines) per application corresponding with the same executions. Fig. 9 shows overhead introduced by

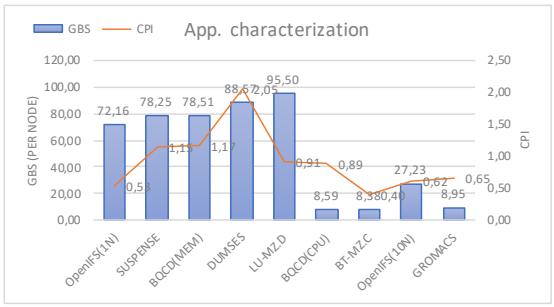

Fig. 7. Application characterization: GBS and CPI

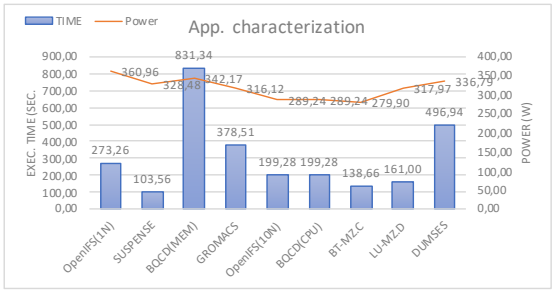

Fig. 8. Application characterization:Execution time and Avg. DC node power

EARL against running the application without EARL, both cases at 2.4GHz. Values higher than 1 means EARL overhead, so as we can see, EARL is not introducing any overhead since some applications has even reported less execution time when running with EARL. This can be explained because

applications suffer minor variations in execution time even running in the same set of nodes.

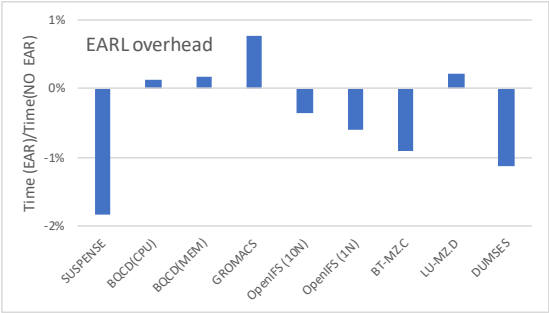

Fig. 9. EARL overhead: Execution time when running with EAR (MONITORING_ONLY) vs Not loading EAR F=2.4GHz

Fig. 10 shows the frequencies selected for the different applications and use cases for the two energy policies. Applications presented in the right side are "cpu" applications and applications in the left side are "memory" applications. `MIN_TIME_TO_SOLUTION` has been executed with a default frequency of 2.1GHz and a `policy_threshold`=75%. `MIN_ENERGY_TO_SOLUTION` has been executed with a default frequency of 2.4GHz and a `policy_threshold`=10%. The `MIN_TIME_TO_SOLUTION` policy maintains "memory" applications running at default frequency whereas "cpu" applications are boosted up to the maximum frequency. `MIN_ENERGY_TO_SOLUTION` shows a clearly different behaviour and not so deterministic in terms of application characteristics but in general CPU frequency for "memory" applications is reduced since the impact on performance is not so significant.

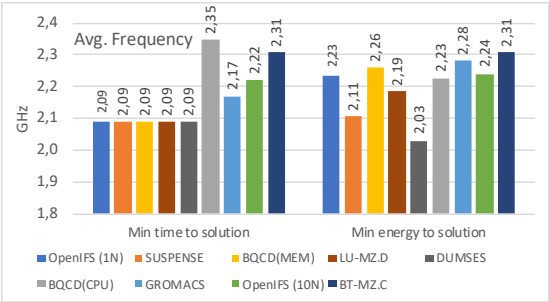

Fig. 10. Frequencies selected when running with MIN_TIME_TO_SOLUTION and MIN_ENERGY_TO_SOLUTION policies

Fig. 11 shows EARL performance evaluation when running the `MIN_TIME_TO_SOLUTION_POLICY`. We have compared time and energy when running with `MIN_TIME_TO_SOLUTION_POLICY` against time and energy when running at constant frequencies of 2.4GHz (nominal) and 2.1GHz (default frequency for `MIN_TIME_TO_SOLUTION_POLICY`). Results show how most of the applications don't suffer a significant slowdown compared to when running at 2.4GHz except OpenOFS(1N) (11% of slowdown) and BQCD(MEM) (4%). The case of BQCD(MEM) is going to be improved in the future work since it presents two internal phases. One of the phases is the memory intensive one and is the one dominating the application behaviour. However, the other one is not so memory intensive and it is a bit delayed. OpenIFS(1N) is also showing a mixed behaviour, since it is reporting a medium/high memory bandwidth but a low CPI.For the rest of the applications there is not slowdown either because they are not taking benefit of high CPU frequencies or because they have been boosted (third/grey bar shows the speedup compared with 2.1GHz). In the case of OpeIFS(10N), even though the application has been boosted up to 2.22Ghz, it has reported a 5% slowdown compared with running the application at 2.4GHz but 11% of speedup compared when running it at 2.1GHz (which is the default frequency). Concerning energy, the not-accelerated applications have reported the bigger benefits as expected. Boosted applications have reported similar energy consumption than the execution at constant 2.4GHz. With this policy, applications reporting a good ration between performance increment vs frequency increment will not be delayed and applications with not enough benefit will be executed at lower frequencies, being more or less delayed with respect the execution at the maximum frequency but reporting clear energy benefits. One important point to remark is the difference between cases corresponding to the same binary: BQCD and OpenIFS. In these cases, EARL has no problem in identifying the runtime characteristics and doing the frequency selection accordingly.

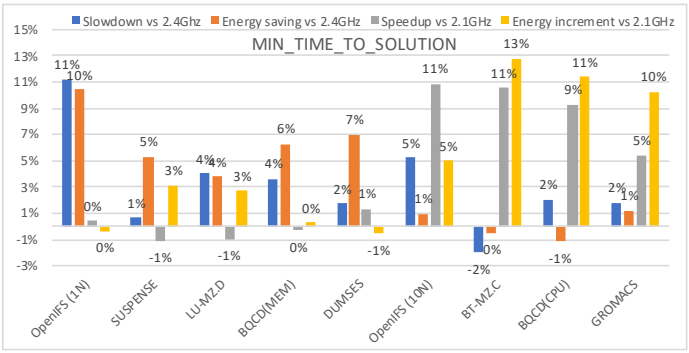

Fig. 11. MIN_TIME_TO_SOLUTION policy evaluation

Fig. 12 shows EARL performance evaluation when running the `MIN_ENERGY_TO_SOLUTION_POLICY`. In the x axis we show the different applications and for each one the slowdown introduced with respect the nominal frequency (2.4GHz) and the energy savings compared with the application executed without EARL at a constant frequency of 2.4GHz. With this policy, there is not a criteria of ratio between performance penalty and energy saving. Absolute values are used for frequency selection. This is because frequencies are not so clearly differentiated between application types. In this experiment

there are two main remarks: in all the cases the performance penalty has been set to 10%, which is reflected in fig. 12, and all the cases have reported energy savings.

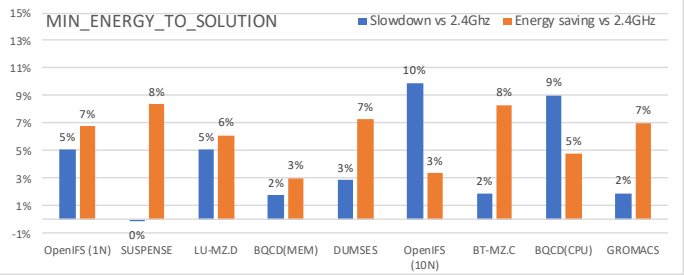

Fig. 12. MIN_ENERGY_TO_SOLUTION policy evaluation

VI. Related work

Some other recent works have been also tackling the power/energy management. Reference [13] presents a complementary approach to EARL. READEX tool suite is a set of tools targeted to help application developers to create an energy efficient version of their applications. It requires much more user intervention since applications much be compiled and tuned to be able to construct the model that will be used later at runtime. Moreover, they rely on external system software, such as the SLURM [15] or HDEEM [14] to compute energy consumption. EAR framework doesn't need any external software to calculate metrics for energy management. Moreover, EAR is not only targeted to energy / power optimization. It is a more general framework offering energy/power control and accounting whereas READEX is limited to a similar scope to EARL.

In [22] authors propose Adagio. A very interesting approach for power management. Adagio is a runtime library targeted to save as much energy as possible minimising the performance penalty. Adagio takes its frequency decisions at runtime and doesn't need user intervention. EAR is similar to Adagio in the sense we target energy optimization at runtime with on line collected information. However, we differ in the approach and the scope of the work. EARL supports two energy policies and our plan for the future work is to specify a clear interface to make it easy the introduction of new policies (such as the power balancing policies used in Adagio). EARL has been evaluated with a significant number of cores and hybrid applications, being ready for production systems. Moreover, our scope coves not only power optimization but also energy accounting and control with the coordination with the EARGM. Reference [16] presents Conductor. It is run-time system that intelligently distributes available power to nodes and cores to improve performance. It is also based on detecting critical paths and using more power in these parts, however, Conductor requires the user to mark the end of the iterative timestep. Conductor targets MPI+OpenMP applications and it can be envision as an extension or improvement of Adagio since it exploits similar concepts in a different context. Conductor exploits the thread level by doing a reconfiguration of the concurrency level and later redistributing the power to speedup the critical path. They assume a context where there is a per-job power bound while we are assuming there are global limits but not per-job limits. Conductor uses a global scheduler after application reconfiguration for power reallocation. It does global syncrhonization for a few time steps, expecting the applications is running so many time step that this overhead will become negligible. We are not applying global synchronisations in EARL to minimize the application interferences. Moreover this approach has a limitation in the power cap for the node, so there is a limit in the amount of power that can be reallocated. Authors reports an overhead of 35 usec. per MPI call. Our algorithm is much more efficient (0,1 usec.) and our experience with the applications evaluated with millions of MPI calls demonstrate that 35usec. is too much for most of them.

GEOPM [23] from Intel also presents an open source framework for power management. GEOPM implements a hierarchical design to support Exascale systems. However, GEOPM doesnt have the technology to dynamically detect the application structure. GEOPM is an extensible framework where new policies can be added at the node level or application (MPI) level. Some of the policies requires application modifications but some others don't have this requirement. However, in this case one additional process is created and metrics used for frequency selection cannot be associated with specific parts of the application structure (for instance outer loops). GEOPM doesn't include the EARL capability of dynamic application characterisation offered by DynAIS to EARL. Energy control in GEOPM is not included as part of GEOPM framework, it offers an API to be used by the resource manager. In the case of EAR, a complete framework is provided. DVFS has been also used in other works to reduce the power consumed by applications. In the context of MPI applications, DVFS have been used extensively inside the MPI library to reduce the power consumed during communication periods [24] [25] [26]. The goal of these proposals was to reduce the power consumed inside the MPI library without introducing a significant performance degradation in the application execution. EAR considers not only communication parts, it takes into account all the code executed by the application, reducing the frequency considering the application as a whole, being able to detect, for instance, memory bounded applications which present good opportunities for energy savings. Power capping has been also targeted at the scheduler and resource manager level, for instance in SLURM [27] or PBS [28] to control the total power consumption. Frequency selection at the cluster was used in the LSF scheduler [29] [30]. In this implementation, frequency selection was statically done at the job submission based on user provided information at job submission and historic information. Relying on user information is a source of errors, limiting the success of the proposal.

VII. Conclusions and future work

This paper presents EAR, an energy management framework offering energy frequency optimisation, energy accounting, and energy/power control. EAR doesn't depend on other system frameworks for a complete energy management solution. The EARL (EAR library) offers energy optimisation by selecting the optimal frequency at runtime with a negligible overhead. EARL detects the application structure on the fly using DynAIS algorithm, offering EARL the ability to self-evaluate its decisions. It includes two energy policies for two different environments but our future work includes extending EARL with additional policies and techniques for power capping and power balance. We will also work on extending DynAIS to provide a fine grain control over the inner loops conforming outer loops. EARL has been evaluated with real applications with more than 1000 cores and showing energy savings up to 10% (DC node energy). EARL provides energy control in coordination with EARGM, a global component monitoring the energy/power consumed in the system and deciding when energy/power settings need to be updated to reduce the current energy/power consumption. Energy/power actions to adapt to new settings as well as frequency selection are taken totally distributed in compute nodes, being ready for system scalability. EAR also includes a complete energy accounting mechanism joining in a single DB energy and performance metrics related to jobs and nodes.

Acknowledgment

Authors want to thank the EAR team Jordi Aneas and Lluis Alonso, Peter Mayes from Lenovo HPC and AI Application Team for his valuable help in doing the EAR installation and tuning in the evaluation platform, the Application Support Department of the Leibniz Supercomputing Center of the Bavarian Academy of Sciences and Humanity for valuable comments and suggestions on designing EAR. Authors also wants to thanks Prof. Uhlmann for allowing us to use SUSPENSE in our evaluation, and Kim Serradell from Earth Sciences deparment for the guidance on how to use OpenIFS

References

[1] BQCD. https://www.rrz.uni-hamburg.de/services/hpc/bqcd
[2] Intel Node Manager. https://www.intel.com/content/www/us/en/data-center/data-center-management/node-manager-general.html
[3] SLURM SPAN plugin. https://slurm.schedmd.com/spank.html
[4] ThinkSystem SD650 Direct Water Cooled Server https://lenovopress.com/lp0636-thinksystem-sd650-direct-water-cooled-server
[5] J.R. Deller, Jr., J. G. Proakis, and J.H. Hansen. 1993. Discrete Time Processing of Speech Signals (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA
[6] F. Freitag, J. Corbalan, and J. Labarta. "A dynamic periodicity detector: Application to speedup computation." Parallel and Distributed Processing Symposium., Proceedings 15th International. IEEE, 2001.
[7] M. Uhlmann, "An immersed boundary method with direct forcing for the simulation of particulate flows,Journal of Computational Physics", Volume 209, Issue 2, 2005,Pages 448-476,ISSN 0021-9991, https://doi.org/10.1016/j.jcp.2005.03.017. http://www.sciencedirect.com/science/article/pii/S0021999105001385
[8] A. Auweter, A. Bode,M. Brehm,L. Brochard,N. Hammer,H. Huber,R. Panda,F. Thomas,T. Wilde . "A case study of energy aware scheduling on super-muc." International Supercomputing Conference. Springer, Cham, 2014
[9] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga. "The NAS parallel benchmarks." The International Journal of Supercomputing Applications 5.3 (1991): 63-73.
[10] H. Feng, R. VanderWijngaart, R. Biswas, C. Mavriplis. "Unstructured Adaptive (UA) NAS Parallel Benchmark. Version 1.0." (2004).
[11] Intel Math kernel libray. https://software.intel.com/en-us/mkl/documentation/code-samples
[12] The stream_mpi benchmark. https://www.cs.virginia.edu/stream/FTP/Code/Versions/stream_mpi.c
[13] READEX project. https://www.readex.eu/
[14] T. Ilsche, R. Schne, J.Schuchart, D. Hackenberg, M. Simon, Yi. Georgiou and W. E. Nagel. "Power Measurement Techniques for Energy-Efficient Computing: Reconciling Scalability, Resolution, and Accuracy" In: Second Workshop on Energy-Aware High Performance Computing (EnA-HPC). 2017. DOI: 10.1007/s00450-018-0392-9
[15] SLURM. https://slurm.schedmd.com/
[16] A. Marathe, P.E. Bailey, D.K. Lowenthal, B. Rountree, M.Schulz, B.R. de Supinski. (2015) A Run-Time System for Power-Constrained HPC Applications. In: Kunkel J., Ludwig T. (eds) High Performance Computing. ISC High Performance 2015.Lecture Notes in Computer Science, vol 9137. Springer, Cham https://e-reports-ext.llnl.gov/pdf/789054.pdf
[17] EAR web page. https://www.bsc.es/research-and-development/software-and-apps/software-list/energy-aware-runtime-ear
[18] EAR user guide. https://gateway.bsc.es:11003/sites/default/files/public/bscw2/content/software-app/technical-documentation/ear_guide.pdf
[19] DUMSES. https://github.com/marcjoos-phd/dumses-hybrid
[20] GROMACS. http://www.gromacs.org/
[21] ECMWF. IFS Documentation CY41R1: IFS documentation. Part III: Dynamics and Numerical Procedures".=https://www.ecmwf.int/sites/default/files/elibrary/2014/9203-part-iii-dynamics-and-numerical-procedures.pdf
[22] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. 2009. Adagio: making DVS practical for complex HPC applications. In Proceedings of the 23rd international conference on Supercomputing (ICS '09). ACM, New York, NY, USA, 460-469. DOI: https://doi.org/10.1145/1542275.1542340
[23] J.Eastep, S.Sylvester,C.Cantalupo,B. Geltz,F. Ardanaz,A.Al-Rawi,K.Livingston,F.Keceli,M.Maiterth,S.Jana. "Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions." International Supercomputing Conference. Springer, Cham, 2017.
[24] A.Venkatesh, A.Vishnu, K.Hamidouche, N. Tallent, D. (DK) Panda, D.Kerbyson, and A. Hoisie "A case for application-oblivious energy-efficient MPI runtime." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2015
[25] M. Etinski, J. Corbalan, J. Labarta, M. Valero and A. Veidenbaum. "Power-aware load balancing of large scale MPI applications." Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE, 2009.
[26] M. Y. Lim, V. W. Freeh and D. K. Lowenthal. "Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs." SC 2006 conference, proceedings of the ACM/IEEE. IEEE, 2006.
[27] SLURM Power Adaptive Computing. https://slurm.schedmd.com/SLUG15/Power_Adaptive_final.pdf
[28] PBSpro Green computing. http://www.pbsworks.com/PBSSolution.aspx?v=1&i=6&n=Green-Computing
[29] R.Bell, L.Brochard, D.DeSotta, R.Panda, F.Thomas. Energy-Aware job scheduling for cluster environments. Patent US 8,527,997 B2. November 2011
[30] L.Brochard, R.Panda, D.DeSota, F.Thomas, and R.H. Bell, Jr. "Power and energy-aware processor scheduling." ACM SIGSOFT Software Engineering Notes. Vol. 36. No. 5. ACM, 2011.
[31] The MariaDB Foundation https://mariadb.org/
[32] PAPI. http://icl.cs.utk.edu/papi/
[33] GNU FreeIPMI https://www.gnu.org/software/freeipmi/