Energy Aware Runtime (EAR) documentation

User guide

This document is part of the Energy Aware Runtime (EAR) framework. It has been created in the context of the BSC-Lenovo Cooperation project.

Contact

BSC: Julita Corbalan julita.corbalan@bsc.es, ear-support@bsc.es

LENOVO: hpchelp@lenovo.com

1 EAR USER GUIDE

1.1 EAR license

- BSC Contact ear-support@bsc.es
- Lenovo contact hpchelp@lenovo.com

1.2 EAR overview

Energy Aware Runtime (EAR) is designed to provide a simple solution for MPI applications to be energy efficient. EAR includes several components but the main one is the EAR library (EARL). This document describes how to use EAR library automatically.

EAR is designed to work independently of any scheduler, however, we provide a SLURM SPANK plugin to make easy the EAR utilization. This document describes main EAR features when running with SLURM support.

EAR utilization in systems with SLURM support installed is very simple. EAR plugin extends srun/sbatch options, sets job configuration before job execution and contacts with EARD[1] to notify job execution. These steps are done transparently to users.

When submitting a job, users can specify the following EAR options (some of them can be limited by the sysadmin):

- power policy: EAR offers two power policies. MIN_ENERGY_TO_SOLUTION and MIN_TIME_TO SOLUTION. There is also a special “policy” called MONITORING_ONLY. This is not a real policy but it can be used by users to collect performance and power metrics. [2]

[1] EARD are EAR daemons running at compute nodes

[2] Energy policies are described in section “EAR policies description”

- power policy threshold: MIN_ENERGY_TO_SOLUTION and MIN_TIME_TO_SOLUTION policies used a threshold to parametrize their behaviour.
- energy tag: a short text “energy tag” characterizing application execution. Some energy-tags have associated a pre-defined power configuration.
- verbose level: from 1..5. Defines the stderr verbosity when running the EAR Library
- mpi distribution: EAR supports Intel MPI and OpenMPI (available if specified at installation time)
- user DB: EAR library collects performance and power metrics during the application execution. When using this option, a csv file per node with metrics collected are generated at application end.

Energy tags are used two folds: they are stored in the DB as part of the accounting information but they can be pre-defined by sysadmins. In this second case, sysadmins can specify which users are allowed to use a given energy tag. When a user sets an energy tag with “authorization”, the predefined configuration is set and EAR library is not loaded.

When submitting a job, there are two possible scenarios: 1) Systems configured with EAR library loaded by default with all the jobs, and 2) systems where EAR library is not loaded by default. EAR library on/off is defined by the sysadmin.

1.3 EAR library on by default

In that case, EAR library is loaded with all the jobs by default. However, that will only affect to MPI jobs since EAR uses the profiling MPI interface to monitor and control dynamically applications. Default power policy settings are defined by the sysadmin and users don’t need to specify any EAR option to be used with EAR. However, depending on EAR configuration, users are allowed to change some of the default settings such as the power policy.

1.4 EAR library off by default

In that case, EAR library is not loaded by default with all the jobs and users must activate it explicitly. This can be done in two ways: by setting the option --ear=on or by using any of the ear options.

1.5 Job submission with ear

EAR options previously presented are specified when doing srun and/or sbatch. EAR options supported with srun/sbatch/salloc are:

Option	Description
--ear=on|off	Enables/disables EAR library
--ear-policy=policy	Selects an energy policy for EAR {policy=MIN_ENERGY_TO_SOLUTION|MIN_TIME_TO_SOLUTION |MONITORING_ONLY}

--ear-cpufreq=frequency	Specifies the start frequency to be used by EAR policy (in KHz)
--ear-policy-th=value	Specifies the ear_threshold to be used by EAR policy {value=[0..1]}
--ear-user-db=file	Specifies the file to save the user applications metrics summary 'file.nodename.csv' file will be created per node. If not defined, these files won't be generated.
--ear-mpi-dist=dist	elects the MPI distribution for compatibility of your application{dist=intel|openmpi|openmpi-fortran}
--ear-verbose=value	Specifies the level of the verbosity {value=[0..5]}; default is 0
--ear-tag=tag	Selects an energy tag
--ear-learning=p_state	Enables the learning phase for a given P_STATE{p_state=[1..n]}

(*) Options in red needs privileges to be used. Option in pink doesn't need privileges but values are limited by EAR configuration.

1.5.1 srun examples

Different examples can be found in folder src/tests/plugin/single_node. Tests from 1 to 100 are srun examples and tests starting at 100 are sbatch examples.

EAR plugin reads srun options and contacts with EARD. Invalid options are filtered to default values, so behavior depends on system configuration.

Executes application with EAR on/off (depending on the configuration) with default values

```
srun -J test -N 1 -n 24 --tasks-per-node=24 application
```

Executes application with EAR on with default values and verbose set to 1

```
srun --ear-verbose=1 -J test -N 1 -n 24 --tasks-per-node=24 application
```

Executes application with EAR on and verbose set to 1. It user is authorized, job will be executed at 2.0GHz as default freq and with power policy set to MIN_TIME_TO_SOLUTION. Otherwise, default values will be applied.

```
srun --ear-cpufreq=2000000 --ear-policy=MIN_TIME_TO_SOLUTION --ear-verbose=1 -J test -N 1 -n 24 --tasks-per-node=24 application
```

Executes application with EAR. If users is authorized to select the “memory-intensive” tag, its application will be executed according to the definition of the tag in the EAR configuration.

```
srun --ear-tag=memory-intensive --ear-verbose=1 -J test -N 1 -n 24 --tasks-per-node=24 application
```

1.5.2 sbatch examples

When using sbatch , EAR options can be specified in the same way. If more than one srun is included in the job submission, EAR options can be inherited from sbatch to the different sruns or can be specifically modified at individuals srun’s. The following example will set the ear verbose mode for all the job steps to 1. First job step will be executed with default settings and second one with MONITORING_ONLY as policy.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -e test.%j.err
#SBATCH -o test.%j.out
#SBTACH --ntasks=24
#SBATCH --tasks-per-node=24
#SBATCH --cpus-per-task=1
#SBATCH --ear-verbose=1

srun  application
srun --ear-policy=MONITORING_ONLY application
```

1.5.3 Running EAR with mpirun (in slurm systems)

When running EAR using mpirun rather than srun, we have to specify the utilization of srun as bootstrap. Otherwise jobs will not go through the SLURM plugin and EAR option will not be recognized. For example, the following example will run application with MIN_TIME_TO_SOLUTION policy.

```
mpirun -n 10 -bootstrap slurm -bootstrap-exec-args=" --ear-policy=MIN_TIME_TO_SOLUTION” application
```

Bootstrap is an Intel® MPI option but not an OpenMPI option. For OpenMPI srun must be used for an automatic EAR support.

1.6 EAR policies description

EAR offers two energy policies: MIN_ENERGY_TO_SOLUTION and MIN_TIME_TO_SOLUTION. There is a third option, MONITORING_ONLY, but it is not a power policy, it is used for application monitoring and CPU frequency is not modified.

The energy policy is selected by setting the --ear-policy=policy option when submitting the job.

- MIN_ENERGY_TO_SOLUTION: The goal of this policy is to minimize the energy consumed with a limit to the performance degradation. The limit in the performance degradation is set in the ear_threshold option. The MIN_ENERGY_TO_SOLUTION policy will select the optimal frequency that minimizes energy enforcing (performance_degradation <= ear_threshold). When executed with MIN_ENERGY_TO_SOLUTION policy, applications starts at nominal frequency.

$$PerformanceDegradation = (T - T_{default})/T_{default}$$

- MIN_TIME_TO_SOLUTION: The goal of this policy is to improve the execution time while guaranteeing a minimum ratio between performance benefit and frequency increment that justifies that energy consumption. The policy uses ear_threshold option (a minimum efficiency is set by the sysadmin). For example, if ear_threashold=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* ear_threshold).

$$PerfGain = (Time - Time_{new})/Time$$

$$FreqGain = (Freq_{new} - Freq)/Freq$$

When executed with MIN_TIME_TO_SOLUTION policy, applications starts at a default predefined frequency lower than nominal (defined at ear.conf[3]). For example, given a system with a nominal frequency of 2.3GHz and default frequency set to 3, an application executed with MIN_TIME_TO_SOLUTION will start with frequency F_i=2.0Ghz (3 p_states less than nominal). When application metrics are computed, the library will compute performance projection for F_{i+1} and will compute the performance_gain as shown in figure 2. If performance gain is greater or equal than ear_threashold, the policy will check with the next performance projection F_{i+2}. If the performance gain computed is less than ear_threashold, the policy will select the last frequency where the performance gain was enough, preventing the waste of energy.

[3] Global EAR configuration file

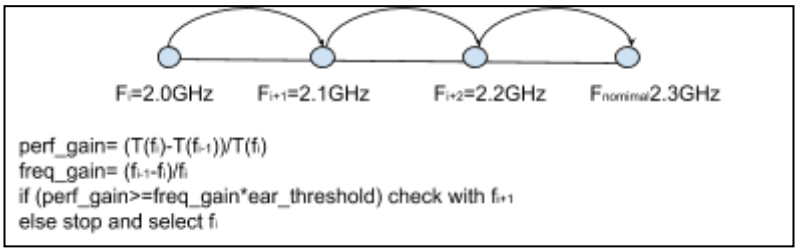


Figure 2: MIN_TIME_TO_SOLUTION uses ear_threashold as the minimum value for the performance gain between between Fi and Fi+1

1.7 Using EAR API in applications

EAR offers an user API for applications. Current EAR version only offers two functions, one to read the accumulated energy and time and another one to compute the difference between two measurements.

- void ear_energy(ulong *energy_mj,ulong *time_ms);
- void ear_energy_diff(ulong ebegin,ulong eend, ulong *ediff, ulong tbegin, ulong tend, ulong *tdiff);

EAR include and library can be found at $EAR_INSTALL_PATH/include and $EAR_INSTALL_PATH/lib respectively. One example can be found at src/test/app_api/app_api_test.c.

```
unsigned long e_mj=0,t_ms=0,e_mj_init,t_ms_init,e_mj_end,t_ms_end=0;
/* Energy reading */
ear_energy(&e_mj_init,&t_ms_init);

/****** DO COMPUTATION *****/
/* Energy reading*/
ear_energy(&e_mj_end,&t_ms_end);
/* Computing the difference */
 ear_energy_diff(e_mj_init,e_mj_end, &e_mj, t_ms_init,t_ms_end,&t_ms);
printf("Time consumed %lu (ms), energy consumed %lu(mJ), Avg power %lf(W)\n",t_ms,e_mj,(double)e_mj/(double)t_ms);
```

2 EAR INSTALLATION

Steps to install EAR are the following ones:

1. Check requirements (libraries and kernel drivers)
2. Building EAR components: configuring, compiling and installing EAR
3. Set the environment: create folders, create DB, configure ear.conf, set EAR plugin (update /etc/slurm/plugstack.conf)

4. Start services → At this point EAR can be used for energy accounting and control, but power policies cannot be applied
5. Execute learning phase and compute coefficients → Once coefficients are computed, power policies can be used [4]

2.1 Hardware requirements

EAR has been optimized for Intel® Skylake architectures. It has been already executed in previous Haswell systems but not intensively tested and many features are not supported.

2.2 Software requirements

EAR requires some third party libraríes and headers to compile and run, in addition to the basic requirements such as the compiler and Autoconf. This is a list of these libraries, and minimum tested versions:

Software	Required / comment	Minimum version
PAPI	Yes / with RAPL support	5.4.3.0
GSL	Yes	1.4
CPUPower	Yes	Kernel 3.10*
FreeIPMI	Yes	1.5.7
SLURM	Just for SLURM plugin (recommended)	17.02.6
MPI	Yes	-
Mysql client	Yes	-

- (*) Depending on the version, may you have to change the name of the library function call (or the parameter).
- We need the developer version of these libraries to compile EAR. When installing from rpm's, you will have to install, at least, the following rpm: gsl-devel, freeipmi-devel, cpupower-devel, and libmysqld-devel

Also, some drivers must be present and loaded in the system:

Driver	File	Kernel version

[4] Coefficients quality are critical for policy performance, we provide several command line tools to check applications executed during the learning phase as well as coefficients

CPUFreq	kernel/drivers/cpufreq/acpi-cpufreq.ko	3.10
Open IPMI	kernel/drivers/char/ipmi/*.ko	3.10

EAR has been compiled with icc and gcc. Latest versions are recommended for performance but the minimum tested versions are 4.8.5 for gcc and 17.0.1 for icc. Performance benefits of EARL can be significantly affected when using old compiler versions.

Since EAR intercepts MPI calls, MPI compiler is also needed. MPI compiler is also used to compile kernels used in the learning phase. Some of the kernels are mpi fortran codes, so a mpi C and fortran installation are also required.

EAR reports information to a MySQL DB. We use MariaDB as MySQL server. EAR assumes a MySQL server has been previously configured in the cluster.

2.3 Building EAR components: configuring, compiling and installing EAR

1. Generate Autoconf's configure program by typing “autoreconf -i”.
2. Compile the library by typing ./configure, make and make install in the root directory. Consider the option of ./configure --PREFIX=<path> if you want to specify the installation path. It could be useful to run ‘./configure --help’ for listing the options details. Configure command looks at default paths to find EAR requirements, however, for specific paths configure options are provided such as “--with-papi=PATH“
3. make & make install

EAR supports the creation of one or two library versions to be loaded with Intel and Openmpi MPI libraries. Specific MPI compilers must be specified (including absolute paths) must be specified in MPICC and OMPICC flags. Specific paths such as etc_ear_path and/or tmp_ear_path can be also specified, as well as the EAR installation path.

> Example using gcc compiler: ./configure --prefix=ear_installation_path --with-papi=papi_path --with-freeipmi=freeipmi_path CC=gcc --with-slurm=/usr TMP=/var/ear --sysconfdir=/etc OMPICC=openmpi_path/bin/mpicc MPICC=default_intel_mpi_path/bin/mpiicc
>
> Example using icc compiler: ./configure --prefix=ear_installation_path --with-papi=papi_path --with-freeipmi=/freeipmi_path CC=icc MPICC=mpiicc CC_FLAGS=-static-intel MPICC_FLAGS=-static-intel --with-slurm=/usr TMP=/var/ear --sysconfdir=/etc OMPICC=openmpi_path/bin/mpicc

Makefile include individual targets for individual components: library.install, commands.install, daemon.install as well as the typical global targets (make install).

Etc files are special files not installed by default. They can be installed using make etc.install. Etc files include different folders. Ear folder inside etc ($EAR_ETC/ear) is supposed to be used “as it is”. Admin can decide to copy or link it at default system paths such as /etc/ear.

- ear: This folder contains ear.conf and some specific subfolders such as "coeffs" for coefficients. Ear components use this folder, so EAR_ETC environment variable must point to it.
- module : This folder contains ear module. It can be useful for some commands since it defines ear installation path, manpath etc. Admin has to copy at modulefiles path.
- slurm : This folder contains a ear.plugstack.conf example
- systemd: This folder contains ear services. Admin has to copy them at services path or create links to EAR services to make it visible to the system. After that, "systemctl daemon-reload" command must be executed

2.3.1 Makefile targets

- make: compiles the project.
- make full: cleans the project and compiles it again.
- install: installs the basic binaries and libraries (the folders bin, lib and sbin).
- make devel.install: install the developer version of the project with additional files, scripts and headers.
- make etc.install: installs just the content of the etc folder.
- make *component*.install: installs just the specified *component* of the project (in case it exists). Supported *components* are: library, commands, common, control, daemon, database_cache, global_manager, metrics, slurm_plugin and tools.
- make clean: removes the compiled files on the project folder.
- make check: runs some tests to check the status of the installation. Sudo may be needed in case some system features require privileges.
- make depend: regenerates the .depend files, which contains the source files and its header dependencies.
- make depend-clean: removes all the .depend files.

2.4 Configure/Set the environment

- EAR folders: EAR uses two [or three] special paths.
  - EAR_TMP=*tmp_ear_path* must be a private folder per compute node. It must have read/write permissions for normal users. Communication files are created here. tmp_ear_path must be created by the admin.
    - For instance: `mkdir /var/ear;chmod ugo +rwx /var/ear`
  - EAR_ETC=*etc_ear_path* must be readable for normal users in all compute nodes. It can be a shared folder in "GPFS" (simple to manage) or replicated data because it is very few data and modified at a very low frequency (ear.conf and coefficients). Coefficients can be installed in a different path specified at configure time in COEFFS flag. Anyway, both ear.conf and coefficients must be readable in all the nodes (compute and "service" nodes).
- MySQL DB: EAR saves data in a MySQL DB server. DB must be created. *ear_create_database* command is provided (MySQL server must be running and root access to the DB is needed)
- Configure ear.conf: ear.conf is an ascii file setting default values and cluster descriptions. An ear.conf is automatically generated based on a ear.conf.in template.

However, sysadmin must include installation details such as hostname details for EAR services, ports, default values, and list of nodes.

- set EAR plugin
  - EAR plugin must be set in /etc/slurm/plugstack.conf. EAR generates an example at *ear_etc_path*/slurm/ear.plugstack.conf
  - EAR plugin code must be specified as well as plugin arguments:
    - Example: required *ear_install_path*/lib/earplug.so prefix=*ear_install_path* sysconfdir=*etc_ear_path* localstatedir=*tmp_ear_path* default=on
  - prefix: sets the path where the installation of EAR files will be placed.
  - sysconfdir: personalizes the path installation specifically for the read only files, such as configuration or coefficient files.
  - localstatedir: personalizes the path installation specifically for the temporal files, such as pipes for inter-process comunications or lock files.
  - docdir: personalizes the path installation specifically for the document files.
  - default=on means EAR library is loaded by default. default=off means EAR library is not loaded by default

2.5 EAR configuration: Ear.conf

ear.conf is a text file describing EAR options cluster description. It must be readable at all compute nodes and at nodes where commands are executed. Lines starting with # are comments. Some of the arguments are optional. A test for ear.conf file can be found at ear_src_path/src/test/functionals/ear_conf. Here it is the list of fields and a basic description.

```
# Parameters
#Services configuration
#Mariadb CONFIGURATION


MariaDBIp=XXX.XXX.XXX.XXX
MariaDBUser=ear_daemon
MariaDBPassw=
MariaDBPort=0
MariaDBDatabase=EAR_DB

#EARGM configuration

GlobalManagerVerbose=1
# Period T1 and Period T2 are specified in seconds T1 (ex. must be less than T2, ex. 10min and 1 month)
GlobalManagerPeriodT1=90
GlobalManagerPeriodT2=259200
#Units can be -=Joules, K=KiloJoules or M=MegaJoules
GlobalManagerUnits=K
GlobalManagerEnergyLimit=550000 ,
GlobalManagerPolicy=MaxEnergy
GlobalManagerHost=hostname
# port were EARGM will be executed
GlobalManagerPort=50000
# Two modes are supported 0=passive 1= pro-active modes
```

```
GlobalManagerMode=0,
# independetly on the mode, a mail can be sent reporting the warning level (and the action taken in automatic mode). nomail means no mail is sent
GlobalManagerMail=nomail
# Thee values must be provided corresponding with DEFCON_L4,DEFCON_L3, and DEFCON_L2 (higer values means PANIC)
GlobalManagerWarningsPerc=85,90,95
# number of "grace" T1 periods before doing a new re-evaluation
GlobalManagerGracePeriods=3

# EARD configuration

# different verbose levels are supported (0..4)
NodeDaemonVerbose=1
#  Power Monitoring Frequency in seconds
NodeDaemonPowermonFreq=60
# 1 means nominal frequency (no turbo)
NodeDaemonMaxPstate=1
# 0 means no turbo frequency
NodeDaemonTurbo=0
NodeDaemonPort=5000
# Send data to MySQL DB
NodeUseDB=1
# Send datat to MySQL using EARDBD (1) or directly to the mysql server (0)
NodeUseEARDBD=1
# Specifies if EARD has to Force frequencies (or not) when EARL is not loaded.
NodeForceFrequencies=1

#EARDBD configuration
# In seconds, time of accumulating data in every aggregation
DBDaemonAggregationTime=60
# In seconds, time between insert the buffered data
DBDaemonInsertionTime=30
# Port where the EARDBD server is listening
DBDaemonPortTCP=4711 , This port is used for main EARDBD
# Port where the EARDBD mirror is listening
DBDaemonPortSecTCP=4712
# Port is used to synchronize the server and mirror
DBDaemonSyncPort=4713
# Memory allocated per process. It means that if there is a server and mirror in a node a double of that value will be allocated. It is expressed in MegaBytes.
DBDaemonMemorySize=120
# The percentage of the memory block used by each type. These types are: mpi, non-mpi and learning applications, loops, energy metrics and aggregations and events, in that order.
DBDaemonMemorySizePerType=40,20,5,24,5,1,5


# EARL configuration
# path where coefficients are installed
CoefficientsDir= (EAR_ETC)/ear/coeffs
# number of levels used by DynAIS algorithm
DynAISLevels=4
# Windows size used by DynAIS, the higher the size the higer the overhead
DynAISWindowSize=500
# Maximum time (in seconds) EAR will wait until a signature is computed. After
# DynaisTimeout seconds, if no signature is computed, EAR will go to periodic mode
DynaisTimeout=30
```

```
# When EAR goes to periodic mode, it will compute the Application signature every "LibraryPeriod" seconds
LibraryPeriod=30
# EAR will check every N mpi calls whether it must go to periodic mode or not
CheckEARModeEvery=1000

# Paths
# this path is used for communitation files, shared memory, etc. It must be PRIVATE
# per compute node and with read/write permissions
TmpDir=/var/ear
#EAR_ETC path where coefficients and configuration are stored. It must be readable in all
# compute nodes (it can be replicated).
EtcDir=/etc/
#  when no DB is installed, metrics are generated in text files.
DataBasePathName=/etc/ear/dbs/dbs.

# Energy policies configuration
# default policy
DefaultPowerPolicy=MIN_TIME_TO_SOLUTION
# list of allowed policies for normal users
#(it is a subset of MONITORING_ONLY, MIN_TIME_TO_SOLUTION, MIN_ENERGY_TO_SOLUTION) ,
SupportedPolicies=MONITORING_ONLY,MIN_TIME_TO_SOLUTION
# specified in the following order:
# MIN_ENERGY_TO_SOLUTION,MIN_TIME_TO_SOLUTION,MONITORING_ONLY
DefaultPstates=1,4,4
# threshold used for MIN_TIME_TO_SOLUTION policy
MinEfficiencyGain=0.7
# threshold used for MIN_ENERGY_TO_SOLUTION policy
MaxPerformancePenalty=0.1
# min time between two energy readings for performance accuracy
MinTimePerformanceAccuracy=10000000

# Security configuration
# Authorized users, groups, and slurm accounts are allowed to change policies, thresholds,
# frequencies etc they are supposed to be admins . A list of users, linux groups, and/or
# SLURM accounts can be provided.

AuthorizedUsers=user1
AuthorizedAccounts=acc1,acc2,acc3
AuthorizedGroups=group1,group2

# List of energy tags and users/groups/SLURM accounts authorized to use each one.
# These energy tags implies pre-defined configurations for applications (EAR library
#  is not loaded)
EnergyTag=memory-intensive pstate=4 users=all groups=group1,group2 accounts=acc1,acc2

# Special nodes configuration
# Describes nodes with some special characteristic such as different default pstates, default # coefficients file, and/ot policy thresholds, only changes must be included

NodeName=nodename_list CPUs=24 DefaultPstates=2,5,5 DefCoefficientsFile=filename MaxPerformancePenalty=def_th MinEfficiencyGain=def_th

# Cluster description
# Nodes are grouped in islands, this section is mandatory since it is used for cluster
```

```
# description more than one line per island must be supported to specify different
# dbip  ports.
# One EARDBD cannot be mirror from more than one EARDBD

Island=0 Nodes=nodename_list DBIP=EARDB_hostname DBSECIP=EARDB_mirror_hostname
```

2.6 Starting EAR components

COMMENT: EAR uses a MariaDB server. The MariaDB server must be started before EAR services are executed.

Three of the EAR components can be started are Linux services. Service files are generated during "configure" process using etc/systemd/*.service.in templates. EAR includes templates for EAR daemon, global manager, and database manager. These service files can be automatically installed (copied) at the etc_ear_path/systemd folder by doing 'make etc.install'. They can be also manually copied at any folder the sysadmin uses as services path.

- eard.service: EAR daemons must be running in all the compute nodes
- eargm.services: EAR global manager, if started, must be running in a single node and it needs DB access
- eardbd.service: EAR database manager, if started, must run in nodes with DB access. EAR supports multiple database managers in a single cluster. EAR database managers and ports must be specified at ear.conf.

Services can be started (stopped,reloaded) on parallel using parallel commands such as pdsh.

Example: sudo pdsh -w nodelist systemctl start eard

2.7 EAR Learning phase

EAR library uses a set of per-node coefficients that must be computed once EAR is installed and each time changes in the hardware can affect performance and/or power. The README file at etc/scripts/learning folder describes how to configure and execute the learning phase in more detail.

The learning phase includes three main steps:

1. Kernels compilation and tuning: EAR includes 7 kernels covering cpu and memory intensive use cases. kernels sources can be found under the kernel folder. They must be adapted to the specific number of cores and with a minimum execution time (aprox. 1-2 minutes per kernel).

2. Execution of the kernels with the range of selected frequencies executed in all the nodes we want to compute coefficients (nodelist must be provided), and the number of runs(per kernel and p_state) to minimize the variances (3 runs is recommended).

3. Computation of coefficients once kernels are executed.

Steps 2 and 3 are automatically done by the script bin/scripts/learning_phase_execute.sh included with EAR. However, the initial kernel tuning must be validated by the sysadmin. EAR includes another script (etc/scripts/learning/learning_phase_compile.sh) to automatically compile and execute the kernels, but the number of cores and the validation of the execution time must be done manually.

Coefficients are generated at $EAR_ETC/ear/coeffs, and saved on folders based on the definition of ear.conf. When reading coefficients, EAR will check file existence in the following order:

- etc_ear_path/coeffs/islandX/coeffs.nodename
- etc_ear_path/coeffs/islandX/default_coeff_for_node (specified at ear.conf), if exists
- etc_ear_path/coeffs/islandX/coeffs.default

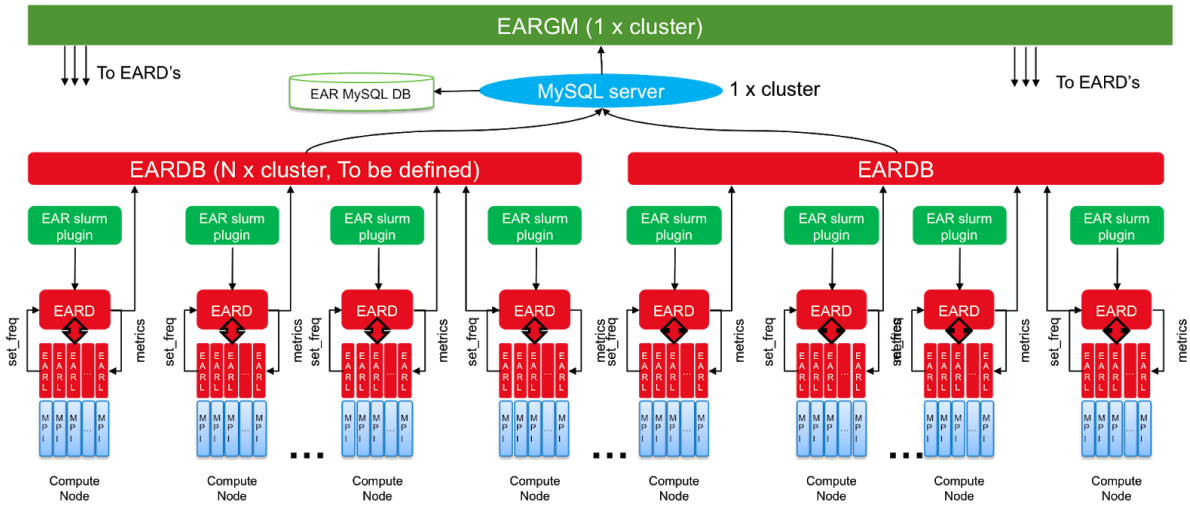
3 EAR COMPONENTS

3.1 EAR overview

EAR has the following components:

- EAR library (loaded with MPI applications using LD_PRELOAD): Offers power policies
- EAR daemon (root): 1 per (compute) node. Offers privileged metrics and energy accounting.
- SLURM plugin (SPANK plugin): Connects with EARD and configure library settings (including LD_PRELOAD)
- EAR Global Manager: 1 process (not mandatory). Controls Global Energy in the cluster [and reacts to warning situations]
- EAR Database Manager: N instances per cluster. Controls DB accesses. Provides buffering and aggregation.
- EAR DB: Available through a MySQL DB. We use MariaDB for that purpose.

The following picture shows main interactions between components

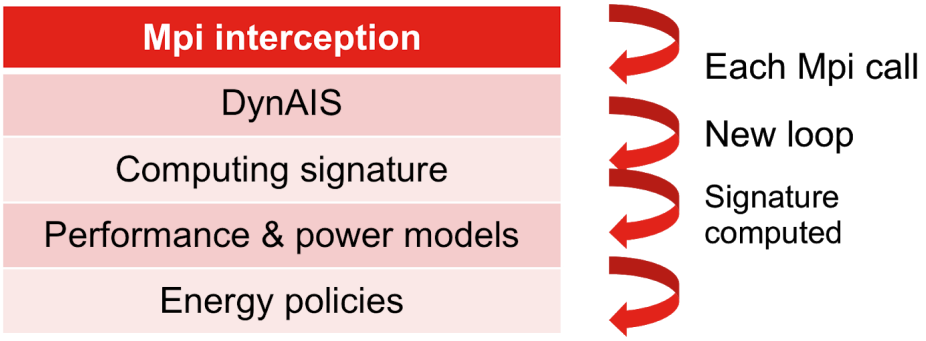


3.2 EAR library (EARL)

The EAR library is the core of the EAR package. The EARL offers a lightweight and simple solution to select the optional frequency for MPI applications at runtime.

EARL is dynamically loaded with applications using the PMPI interface used by many other runtime solutions. The current EARL version only supports with this mechanism but it is under development an API to be inserted in the OpenMPI library.

At runtime, EARL goes through the following phases:



- Automatic detection of application outer loops. This is done by dynamically intercepting MPI calls (using LD_PRELOAD) and invoking DynAIS algorithm, our Dynamic Application Iterative Structure detector algorithm. DynAIS is highly optimized for new Intel architectures reporting low overhead.
- Computation of the application signature. Once DynAIS starts reporting iterations for the outer loop, EAR starts computing the application signature. This signature includes: CPI, iteration time, DC node power and TPI (transactions per instruction). Since DC node power measurement error highly depends on the hardware, EAR automatically detects the hardware characteristics and sets a minimum time to

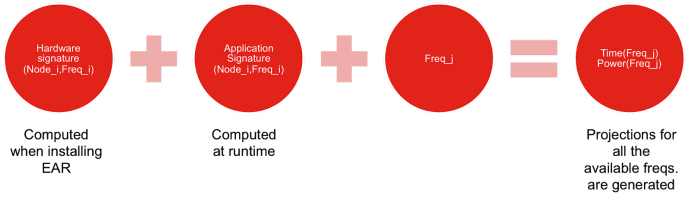
compute the signature in order to minimize the average error.

$$Power(fn) = A(Rf,fn)*Power(Rf) + B(Rf,fn)*TPI(Rf) + C(Rf,fn)$$

$$CPI(fn) = D(Rf,fn)*CPI(Rf) + E(Rf)*TPI(Rf) + F(Rf,fn)$$

$$TIME(fn) = TIME(Rf) * CPI(Rf,fn)/CPI(Rf) * (Rf/fn)$$

- Power and performance projection. EAR has its own performance and power models which uses, as an input, the application signature and the system signature. The system signature is a set of coefficients characterizing each node in the system. They are computed at the learning phase at the EAR configuration time. EAR estimates the power used and execution time of the running application for all the available frequencies in the system.



- Apply the selected power policy. EAR includes two power policies to be selected at runtime: 'minimize time to solution' and 'minimize energy to solution'. These policies are permitted or not by the system administrator. At this point, EAR executes the power policy, using the projections computed in the previous phase, and selects the optimal frequency for this application and this particular run.

3.2.1 Configuration

The EAR Library is based on $EAR_ETC/ear/ear.conf settings when executing in a fully installed environment. Specific settings are available through a shared memory regions initialized by EARD and readable by the EARL. Read the EAR installation guide for more information about EAR configuration.

3.2.2 How to run MPI applications with EARL (with SLURM support)

Even though EAR doesn't need SLURM to be executed, this is the recommended option since it makes totally transparent the execution of jobs with EAR. The SLURM plugin deals

with the configuration of the application and will contact the EARD and EAR Global Manager.

EAR library can be configured by the system administrator to be loaded "by default" or not. When EAR is configured by default, it is not needed to add any option to srun or sbatch. If EAR is disabled by default, users can enable it by using some of the ear options or just adding `--ear=on` , in that case, default configuration will be loaded.

For example:

- ./srun -N2 -n2 --ear=on application → will run application with EAR library with default configuration (defined by sysadmin)
- ./srun -N2 -n2 --ear-policy=MIN_ENERGY_TO_SOLUTION application → will run application with EAR library and will select MIN_ENERGY_TO_SOLUTION power policy. If the user is not allowed to use this policy, the default settings will be applied

  If your application is not an MPI application, the benefits of the EAR library won't be applied. But the SLURM plugin would contact with the daemons in order to monitorize the application metrics and take a decision in case the energy budget is surpassed.

3.3 EAR Daemon (EARD)

The EAR node daemon is the component in charge of providing any kind of services that requires privileged capabilities. Current version is multi-threaded process executed with root privileges. It must be executed 1 instance of EARD per compute node.

EARD provides three basic services, each one covered by one thread:

- Provides privileged metrics such as average frequency, uncore integrated memory controller counters to compute the memory bandwidth, and also energy metrics (DC node, DRAM and package energy).
- Implements a periodic power monitoring service. This service allows EAR package to control the total energy consumed in the system.
- Offers an external API (using sockets) to be notified about new/end jobs in the node and to change the node configuration. These changes will only apply to this node, and not to to all nodes as when changing the ear.conf file.

3.3.1 Requirements

EARD uses CPUPower, FreeIPMI, PAPI (with RAPL component enabled), and mysql. Paths to these libraries must be specified during the installation process when they are not installed in default paths.

3.3.2 Configuration

The EAR Daemon uses the `$(EAR_ETC)/ear/ear.conf` file to be configured. It can be dynamically configured by reloading the service. Read the EAR installation guide for more information about EARD options.

3.3.3 Execution

To execute this component, this `systemctl` command examples are provided:

- `sudo systemctl start eard` to start the EARD service.
- `sudo systemctl stop eard` to stop the EARD service.
- `sudo systemctl reload eard` to force to reload the configuration of the EARD service.

Log messages are generated during the execution. Use journalctl command to see eard message:

- sudo journalctl -u eard -f

3.3.4 Reconfiguration

After executing a "systemctl reload eard" command, not all the EARD options are dynamically updated. The list of updated variables are:

- NodeDaemonMaxPstate
- NodeDaemonVerbose
- Default policy and default policy settings
- NodeDaemonPowermonFreq

To reconfigure other details such as EARD port, coefficients, etc, EARD must be stopped and restarted again.

3.4 EAR Database Manager (EARDBD)

The EAR Database Daemon (EARDBD) caches the records generated by the EARL and EARD in the system and report it to the centralized database. It provides two main services: buffering and aggregation of data. It provides buffering for job information, reducing the number of connections with the DB and the number of messages. It provides also aggregation of periodic metrics reported by EARDs in order to reduce the cost of some queries to control the power and energy consumed by the system.

It is recommended to run several EARDBD daemons if the cluster is big enough, to reduce the number of inserts and connections to the database.

3.4.1 Configuration

The EAR Database Daemon uses the `$(EAR_ETC)/ear/ear.conf` file to be configured. It can be dynamically configured by reloading the service.

Read the EAR installation guide for more information about the options of EARDBD and other components.

3.4.2 Execution

To execute this component, this `systemctl` command examples are provided:

- sudo systemctl start eardbd to start the EARDBD service.
- sudo systemctl stop eardbd to stop the EARDBD service
- sudo systemctl reload eardbd to force to reload the configuration of the EARDBD service.

3.5 EAR Global Manager (EARGM)

EARGM controls the energy consumed in the system following system configuration. It can be configured to work as a system monitoring tool, reporting warning messages, or it can be configured to be pro-active and automatically adapt system settings being coordinated with EAR library. Since EAR library is aware of application characteristics, it can react to the different EARGM warnings levels based on application characteristics and the energy efficiency measured. The combination of EARGM + EAR library makes EAR a Cluster solution for energy management.

3.5.1 Execution

To execute this component, this `systemctl` command examples are provided:

- sudo systemctl start eargmd to start the EARGM service.
- sudo systemctl stop eargmd to stop the EARGM service
- sudo systemctl reload eargmd to force to reload the configuration of the EARGM service.

4 EAR COMMANDS

EAR offers the following commands:

- Commands to analyze data stored in the DB: eacct and ereport
- Commands to control and temporally modify cluster settings: econtrol
- Commands to create/update the DB: ear_create_database, ear_store_database

All the EAR commands read the ear.conf file to determine privileged users.

4.1 Energy accounting (eacct)

Energy accounting (eacct) command shows accounting information stored in the EAR DB for jobs (and step) IDs. eacct command uses EAR configuration to determined users executing the command is normal or privileged user. Normal users can only access to their information. It provides the following filters and options:

- Filters: jobid (+stepid), username, energy-tag, maximum number of jobs to be shown
- Per node or averaged per-job information
- Save metrics in csv file to be post-processed
- Read input from csv files rather than from EAR DB (Mysql)

These options are selected with the following flags provides the following options.

```
Usage: eacct [Optional parameters]
        Optional parameters:
        -h      displays this message
        -u      specifies the user whose applications will be retrieved.
                Only available to privileged users. [default: all users]
        -j      specifies the job id and step id to retrieve with the format [jobid.stepid].
                A user can only retrieve its own jobs unless said user is privileged.
                [default: all jobs]
        -c      specifies the file where the output will be stored in CSV format.
                [default: no file]
        -t      specifies the energy_tag of the jobs that will be retrieved.
                [default: all tags].
        -l      shows the information for each node for each job instead of the
                global statistics for said job.
        -n      specifies the number of jobs to be shown, starting from the most
                 recent one. [default: 20][to get all jobs use -n all]
        -f      specifies the file where the user-database can be found. If this option
                is used, the information will be read from the file and not the database.
```

4.1.1 Example

Job 31191 corresponds with the execution of the bqcd application with 6 job steps. When executing eacct -j 31191 we will get the following output:

```
[user@host EAR]$ eacct -j 31191
JOBID     USER   APP      POLICY NODES FREQ(GHz) TIME(s)   POWER(Watts)  GBS      CPI      ENERGY (J)
  31191.4 user bqcd_cpu      ME    50     2.27      398.38      229.09        4.26     1.00     4563306.92
  31191.3 user bqcd_cpu      ME    50     2.28      394.89      230.84        4.30     0.98     4557703.38
  31191.2 user bqcd_cpu      MO    50     2.38      316.31      272.27        5.35     0.92     4306123.40
  31191.1  user bqcd_cpu     MO    50     2.38      319.97       271.79       5.29     0.92     4348172.40
  31191.0 user bqcd_cpu      MO    50     2.38      317.04      269.67        5.34     0.91     4274739.44
```

Columns shown are: job id.stepid, username, application name, policy (ME=MIN_ENERGY_TO_SOLUTION, MT=MIN_TIME_TO_SOLUTION, MO=MONITORING_ONLY, NP=No Policy (no EARL)), number of nodes, average frequency, execution time, average power, GBs, Cycles per instruction (CP), and energy.

When using -l option, nodename is also reported.

4.2 Energy report (ereport)

Energy report (ereport) is a command that generates reports from the accounting data for nodes. It is oriented to analyze the energy consumption for a given period of time with some additional criteria such as nodename or username.

```
Usage: ereport [options]
Options are as follows:
        -s start_time           indicates the start of the period from which the
                energy consumed will be computed.
                Format: YYYY-MM-DD. Default 1970-01-01.
        -e end_time             indicates the end of the period from which
                the energy consumed will be computed.
                Format: YYYY-MM-DD. Default: current time.
        -n node_name |all       indicates from which node the energy will be
                computed. Default: none (all nodes computed)
                 'all' option shows all users individually, not aggregated.
        -u user_name |all       requests the energy consumed by a user in the
                selected period of time. Default: none (all users computed).
                 'all' option shows all users individually, not aggregated.
        -t energy_tag|all       requests the energy consumed by energy tag in
                the selected period of time. Default: none (all tags computed).
                 'all' option shows all tags individually, not aggregated.
        -h                      shows this message.
```

For instance, if executing the following command line we the output:

```
[user@host EAR]$ ereport -n all -s 2018-09-18
   Energy (J)          Node     Avg. Power
     20668697         node1          146
     20305667         node2          144
     20435720         node3          145
     20050422         node4          142
     20384664         node5          144
     20432626         node6          145
     18029624         node7          128
```

4.3 Energy control (econtrol)

Energy control command (econtrol) connects with EARDs running in compute nodes and temporally modifies their configuration. This changes are not reflected on ear.conf, so they are lost when reloading the system.

```
Usage: econtrol [options]
        --set-freq        newfreq       sets the frequency of all nodes to the requested one
        --set-def-freq  newfreq         sets the default frequency
        --set-max-freq newfreq          sets the maximum frequency
        --inc-th          in_th         increases the threshold for all nodes by inc_th       -
        --red-def-freq num_steps        reduces the default frequency in num_steps pstates
        --restore-conf                  reload the configuration (ear.conf)
        --ping[=nodename]               pings all nodes (or just the selected one)
                                        to check whether the nodes are up or not
```

4.4 Create Database (ear_create_database)

Create database command (ear_create_database) uses MySQL's root user to create the database and tables used by EAR. The database's name and IP are taken from ear.conf file, as well as EAR's default user which will be created by the command too.

```
Usage: ear_create_database [options]
        -p      a password prompt will appear for the admin to introduce root's MySQL password.
```

4.5 Compute model coefficients (compute_coefficients)

After executing the kernels during the learning phase coefficients for the performance and power models must be calculated for each node. EAR includes a command line program to automatically compute coefficients (compute_coefficients) . This program is designed to be executed in the same node where coefficients are computed since the frequency list is automatically detected. It can be also "remotely" executed (passing nodename as argument) but then it is assumed the range of frequencies is the same than the corresponding node.

Compute_coefficients accepts as options the coefficients_path where results will be stored, the minimum frequency to be considered and, optionally, the nodename (by default hostname is used). The command reads the ear.conf file to get DB connection details and nodes configurations (islands).

```
Usage:compute_coefficients coefficients_path min_freq [nodename]
```

4.6 Other EAR commands

EAR includes many other commands mainly used for validating results. It includes, for instance, a program to report in text mode coefficients for a given node (coefficients_to_csv) , or many tests with basic functionality. These programs can be mainly found under src/tools and src/tests. Most relevant ones are:

- compute_coefficients: Computes the node coefficients for a given node. It must be executed in compute node after the execution of the kernels.
- ear_lp_validation: Read kernels metrics executed in the learning phase from the DB and executes different checks to detect the quality of the runs
- get_ear_events: reports EARL events reported to the DB
- show_coefficients: Shows in the stdout coefficients stored in a given file