

Energy Aware Runtime (EAR) package provides an energy management framework for supercomputers. EAR contains different components, all together provide three main services:



1. An **easy-to-use and lightweight optimization service** to automatically select the optimal CPU frequency according to the application and the node characteristics. This service is provided by two components: the EAR library (**EARL**) and the EAR daemon (**EARD**). EARL is a smart component which is loaded next to the application, intercepting MPI calls and selecting the CPU frequency based on the application behaviour on the fly. The library is loaded automatically through the EAR Loader (**EARLO**) and SLURM plugin (**EARPLUG**).
2. A complete **energy and performance accounting and monitoring system** based on SQL database (MariaDB and PostgreSQL are supported). The energy accounting system is configurable in terms of application details and update frequency. The EAR database daemon (**EARDBD**) is used to cache those metrics prior to DB insertions.
3. A **global energy management** to monitor and control the energy consumed in the system through the EAR global manager daemon (**EARGMD**). This control is configurable, it can dynamically adapt policy settings based on global energy limits or just offer global cluster monitoring.

Visit the architecture section for a detailed description of each of these components of EAR.

License

EAR is a open source software and it is licensed under both the BSD-3 license for individual/non-commercial use and EPL-1.0 license for commercial use. Full text of both licenses can be found in COPYING.BSD and COPYING.EPL files.

Contact: ear-support@bsc.es

Running applications with EAR

With EAR's SLURM plugin, running an application with EAR is as easy as submitting a job with either `srun`, `sbatch` or `mpirun` with SLURM. There are multiple configuration settings that can be set to customize EAR's behaviour which are explained below, as well as examples on how to run applications with each method.

For other schedulers a simple prolog/epilog command can be created to provide transparent job submission with EAR and default configuration.

Job submission with EAR and SLURM

The following EAR options can be specified when running `srun` and/or `sbatch`, and are supported with `srun / sbatch / salloc`:

Options	Description
<code>--ear=on/off(**)</code>	Enables/disables EAR library loading with this job.
<code>--ear-policy=policy</code>	Selects an energy policy for EAR. See the Policies page for more info
<code>--ear-cpufreq=frequency(*)</code>	Specifies the starting frequency to be used by the chosen EAR policy (in KHz).
<code>--ear-policy-th=value(*)</code>	Specifies the <code>ear_threshold</code> to be used by the chosen EAR policy { <code>value=[0...1]</code> }.
<code>--ear-user-db=file</code>	Specifies the files where the user applications' metrics summary will be stored {'file.nodename.csv'}. If not defined, these files will not be created.
<code>--ear-verbose=value</code>	Specifies the level of verbosity { <code>value=[0...1]</code> }; the default is 0.
<code>--ear-tag=tag</code>	Selects an energy tag.
<code>--ear-learning=p_state(*)</code>	Enables the learning phase for a given P_STATE { <code>p_state=[1...n]</code> }.

For more information consult `srun --help` output or see configuration options sections for more detailed description.

(*) Option requires *ear privileges* to be used. (**) Does not require *ear privileges* but values might be limited by EAR configuration.

GPU support

EAR version 3.4 and upwards supports GPU monitoring for NVIDIA devices from the point of view of the application and node monitoring. GPU frequency optimization is not yet supported. Authorized users can ask for a specific GPU frequency by setting the `SLURM_EAR_GPU_DEF_FREQ` environment variable. Only one frequency for all GPUs is now supported.

EAR library loading

EAR uses the EAR loader to automatically select the EAR optimization library version. This optimization library is automatically loaded when either an MPI, OpenMP, MKL or CUDA application is detected. Application identification is done based on symbols detection. I doesn't work for static symbols.

MPI versions supported

When using `sbatch/srun` or `salloc`, Intel MPI and OpenMPI are 100% supported. When using `mpi` commands to start applications (`mpirun`, `mpiexec.hydra`, etc.), there are minor differences explained in examples below.

Examples

`srun` examples

EAR plugin reads `srun` options and contacts with EARD. Invalid options are filtered to default values, so behaviour will depend on system configuration.

- Executes an 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 an application with EAR on with default values (policy, default frequency, etc.) and verbose set to 1:

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

EARL verbose messages are generated in the stderr. For jobs using more than 2 or 3 nodes messages can be overwritten. If the user wants to have EARL messages in a file the SLURM_EARL_VERBOSE_PATH environment variable must be set with a folder name. One file per node will be generated with EARL messages.

- Executes an application with EAR on and verbose set to 1. If user is authorised, job will be executed at 2.0GHz as default frequency and with power policy set to monitoring. Otherwise, default values will be applied:

```
srun --ear-cpufreq=2000000 --ear-policy=monitoring --ear-verbose=1 -J test -N 1 -n 24 --task
```

- Executes an application with EAR. If user is authorised to select the “memory-intensive” tag, the 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 appli
```

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 `srun`s or specifically modified in each individual `srun`.

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 as policy.

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

srun application
srun --ear-policy=monitoring application
```

Running EAR with `mpirun` (in SLURM systems)

INTEL MPI

When running EAR with `mpirun` rather than `srun`, we have to specify the utilisation of `srun` as bootstrap. Otherwise jobs will not go through the SLURM plugin and any EAR options will not be recognised. The API depends on Intel version. Versions prior to 2018 use two `mpirun` arguments to specify the bootstrap and extra SLURM flags (to be passed to SLURM).

The following example will run application with `min_time_to_solution` policy:

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

Version 2019 and newer offers two environment variables rather than `mpirun` arguments.

```
export I_MPI_HYDRA_BOOTSTRAP=slurm
export I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS="--ear-policy=monitoring --ear-verbose=1"
mpiexec.hydra -n 10 application
```

OPENMPI

Bootstrap is an Intel® MPI option but not an OpenMPI option. For OpenMPI `srun` must be used for an automatic EAR support, or use the `erun` program explained below.

ERUN

ERUN is a program that simulates all the SLURM and EAR SLURM Plugin pipeline. It comes with the EAR package and is compiled automatically. You can find it in `bin` folder in your installation path. It must be used when a set of nodes does not have SLURM installed or when using OpenMPI `mpirun` which does not contact with SLURM. You can launch ERUN instead of directly run your application:

```
mpirun -n 4 /path/to/erun --program="hostname --alias"
```

In this example, MPIRUN would run 4 ERUN processes. Then, ERUN would launch the application `hostname` with its `alias` parameter. You can use as many parameters as you want but the semicolons have to cover all the parameters in case there are more than just the program name. ERUN would simulate in the remote node both the local and remote pipelines for all created processes. It has an internal system to avoid repeating functions that are executed just one time per job or node, like SLURM does with its plugins.

```
> erun --help
```

This is the list of ERUN parameters:

Usage: ./erun [OPTIONS]

Options:

```
--job-id=<arg>    Set the JOB_ID.  
--nodes=<arg>     Sets the number of nodes.  
--program=<arg>   Sets the program to run.  
--clean           Removes the internal files.
```

SLURM options:

...

The `--job-id` and `--nodes` parameters create the environment variables that SLURM would have created automatically, because it is possible that your application make use of them. The `--clean` option removes the temporal files created to synchronize all ERUN processes.

Also you have to load the EAR environment module or define its environment variables in your environment or script:

Variable	Parameter
EAR_INSTALL_PATH=\<path>	prefix=\<path>
EAR_TMP=\<path>	localstatedir=\<path>
EAR_ETC=\<path>	sysconfdir=\<path>
EAR_DEFAULT=\<on/off>	default=\<on/off>

Lastly, the typical SLURM parameters can be passed to ERUN in the same way they were written to SRUN or SBATCH. In example:

```
mpirun -n 4 /path/to/erun --program="myapp" --ear-policy=monitoring --ear-verbose=2
```

User commands

The only command available to users is `earct`. With `earct` a user can see their previously executed jobs with the information that EAR monitors (time, average power, number of nodes and average frequency among others) and also can use several options to manipulate said output. Some data will not be available if a job is not executed with EARL.

Note that a user can only see their own applications/jobs unless they are a privileged user and specified as such in the `ear.conf` configuration file.

For more information, check its Commands section.

Using the EAR API

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

- `int ear_connect()`
- `int ear_energy(unsigned long *energy_mj, unsigned long *time_ms)`
- `void ear_energy_diff(unsigned long ebegin, unsigned long eend, unsigned long *ediff, unsigned long tbegin, unsigned long tend, unsigned long *tdiff)`
- `int ear_set_gpufreq(int gpu_id, unsigned long gpufreq)`
- `int ear_set_gpufreq_list(int num_gpus, unsigned long *gpufreqlist)`
- `void ear_disconnect()`

EAR's header file and library can be found at `$EAR_INSTALL_PATH/include/ear.h` and `$EAR_INSTALL_PATH/lib/libEAR_api.so` respectively. The following example reports the energy, time, and average power during that time for a simple loop including a `sleep(5)` .

```

#include <ear.h>

int main(int argc, char *argv[])
{
    unsigned long e_mj=0, t_ms=0, e_mj_init, t_ms_init, e_mj_end, t_ms_end=0;
    unsigned long ej, emj, ts, tms, os, oms;
    unsigned long ej_e, emj_e, ts_e, tms_e, os_e, oms_e;
    int i=0;
    struct tm *tstamp, *tstamp2, *tstamp3, *tstamp4;
    char s[128], s2[128], s3[128], s4[128];

    /* Connecting with ear */
    if (ear_connect() != EAR_SUCCESS)
    {
        printf("error connecting eard\n");
        exit(1);
    }

    /* Reading energy */
    if (ear_energy(&e_mj_init, &t_ms_init) != EAR_SUCCESS)
    {
        printf("Error in ear_energy\n");
    }
    while(i<5)
    {
        sleep(5);

        /* READING ENERGY */
        if (ear_energy(&e_mj_end, &t_ms_end) != EAR_SUCCESS)
        {
            printf("Error in ear_energy\n");
        }
        else
        {
            ts=t_ms_init/1000;
            ts_e=t_ms_end/1000;
            tstamp=localtime((time_t *)&ts);
            strftime(s, sizeof(s), "%c", tstamp);
            tstamp2=localtime((time_t *)&ts_e);
            strftime(s2, sizeof(s), "%c", tstamp2);

            printf("Start time %s End time %s\n", s, s2);
            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),\n",
                Avg power %lf(W)\n", t_ms, e_mj, (double)e_mj/(double)t_ms);
            e_mj_init=e_mj_end;
            t_ms_init=t_ms_end;
        }
        i++;
    }
}

```

```

    ear_disconnect();
}

```

EAR offers three energy policies plugins: `min_energy`, `min_time` and `monitoring`. The last one is not a power policy, is used just for application monitoring where CPU frequency is not modified.

The energy policy is selected by setting the `--ear-policy=policy` option when submitting a SLURM job. A policy parameter, which is a particular value or threshold depending on the policy, can be set using the flag `--ear-policy-th=value`. Its default value is defined in the configuration file, for more information check the configuration page for more information.

Plugin `min_energy`

The goal of this policy is to minimise the energy consumed with a limit to the performance degradation. This limit is set in the SLURM `--ear-policy-th` option or the configuration file. The `min_energy` policy will select the optimal frequency that minimizes energy enforcing (performance degradation \leq parameter). When executing with this policy, applications start at default frequency (specified at `ear.conf`).

$$\text{PerfDegr} = (\text{CurrTime} - \text{PrevTime}) / (\text{PrevTime})$$

Plugin `min_time`

The goal of this policy is to improve the execution time while guaranteeing a minimum ratio between performance benefit and frequency increment that justifies the increased energy consumption from this frequency increment. The policy uses the SLURM parameter option mentioned above as a minimum efficiency threshold.

Example: if `--ear-policy-th=0.75`, EAR will prevent scaling to upper frequencies if the ratio between performance gain and frequency gain do not improve at least 75% ($\text{PerfGain} \geq (\text{FreqGain} * \text{threshold})$).

$$\text{PerfGain} = (\text{PrevTime} - \text{CurrTime}) / \text{PrevTime}$$

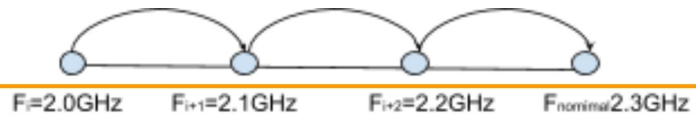
$$\text{FreqGain} = (\text{CurFreq} - \text{PrevFreq}) / \text{PrevFreq}$$

When launched with `min_time` policy, applications start at a default frequency (defined at `ear.conf`). Check the configuration page for more information.

Example: given a system with a nominal frequency of 2.3GHz and default `P_STATE` set to 3, an application executed with `min_time` will start with frequency $F[i] = 2.0\text{GHz}$ (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 the Figure 1. If performance gain is greater or equal than threshold, the policy will check with the next performance projection $F[i+2]$. If the performance gain computed is less than threshold, the policy will select the last frequency where the performance gain was enough, preventing the waste of energy.

Figure 1: `min_time` uses threshold as the minimum value for the performance gain between $F[i]$ and $F[i+1]$.

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/clean the DB: `edb_create` and `edb_clean_pm`

All these commands read the EAR configuration file (`ear.conf`) to determine if the user is an authorized (or not user). Root is a special case, it doesn't need to be included in the list of authorized users. Some options are disabled when the user is not authorized.

Energy Account (`eacct`)

The `eacct` command shows accounting information stored in the EAR DB for jobs (and step) IDs. The command uses EAR's configuration file to determine if the user running it is privileged or not, as non-privileged users can only access their information. It provides the following options.

Usage: `eacct` [Optional parameters]

Optional parameters:

- h displays this message
- v displays current EAR **version**
- u specifies the user whose applications will be retrieved. Only available to
- 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 privi
- a specifies the application names that will be retrieved. [default: all app_i
- c specifies the **file** where the output will be stored in CSV **format**. [default:
- t specifies the energy_tag of the jobs that will be retrieved. [default: all
- l shows the information **for each node for each job** instead of the **global stat**
- x shows the **last** EAR events. Nodes, job ids, and step ids can be specified as
- m prints power signatures regardless of whether mpi signatures are available
- r shows the EAR loop signatures. Nodes, job ids, and step ids can be specifie
- n specifies the **number** of jobs to be shown, starting **from** the most recent one
- f specifies the **file** where the user-database can be found. If this option is
- b verbose mode **for** debugging purposes

Example

The basic usage of `eacct` retrieves the last 20 applications (by default) of the user executing it. If a user is privileged, they may see all users applications. The default behaviour shows data from each job-step, aggregating the values from each node in said job-step. If using SLURM as a job manager, a sb (sbatch) job-step is created with the data from the entire execution. A specific job may be specified with -j:

```
[user@host EAR]$ eacct -j 175966
```

JOB-STEP	USER	APPLICATION	POLICY	NODES	AVG/DEF/IMC(GHz)	TIME(s)	POWER(W)	GBS
175966-sb	user	afid	NP	2	2.97/3.00/---	3660.00	381.51	---
175966-2	user	afid	MO	2	2.97/3.00/2.39	1205.26	413.02	146.
175966-1	user	afid	MT	2	2.62/2.60/2.37	1234.41	369.90	142.
175966-0	user	afid	ME	2	2.71/3.00/2.19	1203.33	364.60	146.

For node-specific information, the -l option provides detailed accounting of each individual node:

```
[user@host EAR]$ eacct -j 175966 -l
```

JOB-STEP	NODE ID	USER ID	APPLICATION	AVG-F/IMC-F	TIME(s)	POWER(s)	GBS
175966-sb	cmp2506	user	afid	2.97/---	3660.00	388.79	---
175966-sb	cmp2507	user	afid	2.97/---	3660.00	374.22	---
175966-2	cmp2506	user	afid	2.97/2.39	1205.27	423.81	146.0
175966-2	cmp2507	user	afid	2.97/2.39	1205.26	402.22	146.3
175966-1	cmp2506	user	afid	2.58/2.38	1234.46	374.14	142.5
175966-1	cmp2507	user	afid	2.67/2.37	1234.35	365.67	142.7
175966-0	cmp2506	user	afid	2.71/2.19	1203.32	371.76	146.2
175966-0	cmp2507	user	afid	2.71/2.19	1203.35	357.44	146.2

For runtime data (EAR loops) one may retrieve them with -r. Both job_id and step_id filtering works:

```
[user@host EAR]$ eacct -j 175966.1 -r
```

JOB-STEP	NODE ID	ITER.	POWER(W)	GBS	CPI	GFLOPS/W	TIME(s)	AVG_F	IMC_F	I/O
175966-1	cmp2506	21	360.6	115.8	0.838	0.086	1.001	2.58	2.30	0.
175966-1	cmp2507	21	333.7	118.4	0.849	0.081	1.001	2.58	2.32	0.
175966-1	cmp2506	31	388.6	142.3	1.010	0.121	1.113	2.58	2.38	0.
175966-1	cmp2507	31	362.8	142.8	1.035	0.130	1.113	2.59	2.37	0.
175966-1	cmp2506	41	383.3	143.2	1.034	0.124	1.114	2.58	2.38	0.

To easily transfer eacct's output, -c option saves it in .csv format. Both aggregated and detailed accountings are available, as well as filtering:

```
[user@host EAR]$ eacct -j 175966 -c test.csv
```

Successfully written applications to csv. Only applications with EARL will have its informatio

```
[user@host EAR]$ eacct -j 175966.1 -c -l test.csv
```

Successfully written applications to csv. Only applications with EARL will have its informatio

Energy report (ereport)

The ereport command creates reports from the energy accounting data from nodes stored in the EAR DB. It is intended to use for energy consumption analysis over a set period of time, with some additional (optional) criteria such as node name or username.

Usage: ereport [options]

Options are as follows:

-s start_time	indicates the start of the period from which the energy consumed w
-e end_time	indicates the end of the period from which the energy consumed wil
-n node_name all	indicates from which node the energy will be computed. Default: no 'all' option shows all users individually, not aggrega
-u user_name all	requests the energy consumed by a user in the selected period of t 'all' option shows all users individually, not aggrega
-t energy_tag all	requests the energy consumed by energy tag in the selected period 'all' option shows all tags individually, not aggregat
-i earbdb_name all	indicates from which earbdb (island) the energy will be computed. 'all' option shows all earbdbs individually, not aggre
-g	shows the contents of EAR's database Global_energy table. The defa This option can only be modified with -s, not -e
-x	shows the daemon events from -s to -e. If no time frame is specifi
-v	shows current EAR version.
-h	shows this message.

Examples

The following example uses the 'all' nodes option to display information for each node, as well as a start_time so it will give the accumulated energy from that moment until the current time.

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

This example filters by EARDBD host (one per island typically) instead:

```
[user@host EAR]$ ereport -s 2019-05-19 -i all
Energy (J)      Node
9356791387      island1
30475201705     island2
37814151095     island3
28573716711     island4
29700149501     island5
26342209716     island6
```

And to see the state of the cluster's energy budget (set by the sysadmin) you can use the following:

```
[user@host EAR]$ ereport -g
```

Energy%	Warning	lvl	Timestamp	INC	th	p_state	ENERGY T1	ENERGY T2
111.486		100	2019-05-22 10:31:34		0	100	893	1011400
111.492		100	2019-05-22 10:21:34		0	100	859	1011456
111.501		100	2019-05-22 10:11:34		0	100	862	1011533
111.514		100	2019-05-22 10:01:34		0	100	842	1011658
111.532		100	2019-05-22 09:51:34		0	100	828	1011817
111.554		0	2019-05-22 09:41:34		0	0	837	1012019

Energy control (econtrol)

The `econtrol` command modifies cluster settings (temporally) related to power policy settings. These options are sent to all the nodes in the cluster.

NOTE: Any changes done with `econtrol` will not be reflected in `ear.conf` and thus will be lost when reloading the system.

Usage: `econtrol [options]`

<code>--status</code>		->requests the current status for all nodes. The on power, IP address and policy configuration . A list responding is provided with their hostnames and IP
<code>--type</code>	<code>[status_type]</code>	->status=node_name retrieves the status of that node ->specifies what type of status will be requested: policy, full (hardware+policy), app_node, app_maste
<code>--set-freq</code>	<code>[newfreq]</code>	->sets the frequency of all nodes to the requested
<code>--set-def-freq</code>	<code>[newfreq] [pol_name]</code>	->sets the default frequency for the selected polic
<code>--set-max-freq</code>	<code>[newfreq]</code>	->sets the maximum frequency
<code>--set-powercap</code>	<code>[new_cap]</code>	->sets the powercap of all nodes to the given value after the value to only target said node.
<code>--restore-conf</code>		->restores the configuration for all nodes
<code>--active-only</code>		->supresses inactive nodes from the output in hardw
<code>--health-check</code>		->checks all EARDs and EARDBDs for errors and print
<code>--mail [address]</code>		->sends the output of the program to address.
<code>--ping</code>		->pings all nodes to check whether the nodes are up <code>--ping=node_name</code> pings that node individually.
<code>--version</code>		->displays current EAR version.
<code>--help</code>		->displays this message.

`econtrol` 's status is a useful tool to monitor the nodes in a cluster. The most basic usage is the hardware status (default type) which shows basic information of all the nodes.

```
[user@login]$ econtrol --status
```

hostname	power	temp	freq	job_id	stepid
node2	278	66C	2.59	6878	0
node3	274	57C	2.59	6878	0
node4	52	31C	1.69	0	0

```
INACTIVE NODES
node1 192.0.0.1
```

The application status type can be used to retrieve all currently running jobs in the cluster. `app_master` gives a summary of all the running applications while `app_node` gives detailed information of each node currently running a job.

```
[user@login]$ econtrol --status --type=app_master
```

Job-Step	Nodes	DC power	CPI	GBS	Gflops	Time	Avg Freq
6878-0	2	280.13	0.37	24.39	137.57	54.00	2.59

```
[user@login]$ econtrol --status --type=app_node
```

Node id	Job-Step	M-Rank	DC power	CPI	GBS	Gflops	Time	Avg Freq
node2	6878-0	0	280.13	0.37	24.39	137.57	56.00	2.59
node3	6878-0	1	245.44	0.37	24.29	136.40	56.00	2.59

Database commands

edb_create

Creates the EAR DB used for accounting and for the global energy control. Requires root access to the MySQL server. It reads the `ear.conf` to get connection details (server IP and port), DB name (which may or may not have been previously created) and EAR's default users (which will be created or altered to have the necessary privileges on EAR's database).

Example

```
Usage:edb_create [options]
  -p      Specify the password for MySQL's root user.
  -o      Outputs the commands that would run.
  -r      Runs the program. If '-o' this option will be override.
  -h      Shows this message.
```

edb_clean_pm

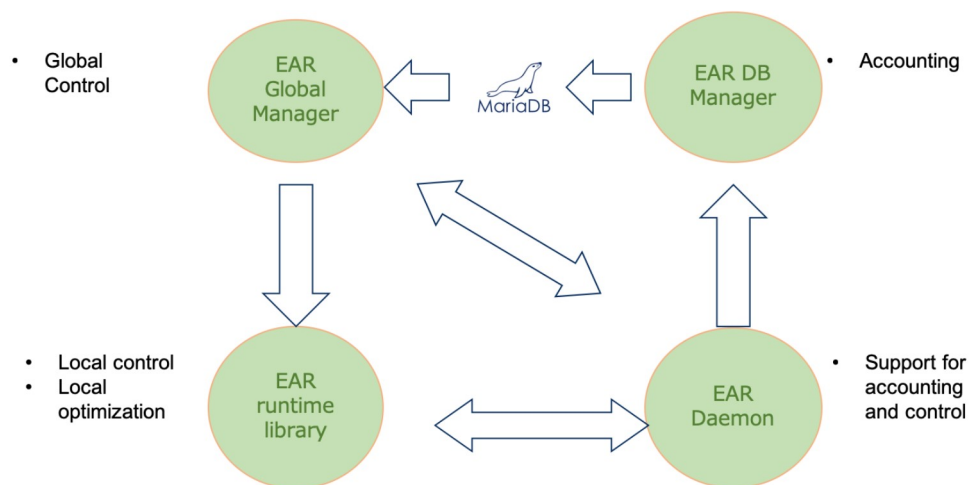
Cleans periodic metrics from the database. Useful to reduce the size of EAR's database. For more information check the FAQs regarding MySQL size management

Components

EAR is composed of five main components:

- Node Manager (EARD). The Node Manager must have root access to the node where it will be running.
- Database Manager (EARDBD). The database manager requires access to the DB server (we support MariaDB and Postgress). Documentation for Postgress is still under development.
- Global Manager (EARGM). The global manager needs access to all node managers in the cluster as well as access to database.
- Library (EARL)
- SLURM plugin

The following image shows the main interactions between components:



Quick Installation Guide

This section provides a, summed up, step by step installation and execution guide for EAR. For a more in depth explanation of the necessary steps see the [Installation from source](#) or [Installation from RPM](#), following the [Configuration](#) and [Execution](#) guides, or contact us at ear-support@bsc.es

Requirements

- To install EAR from sources, the following libraries and environments are needed: C compiler, MPI compiler and library if MPI version is generated, *mysqlclient* for mariaDB or *postgresql* library. *libGSL* is needed for coefficient computations
- To install EAR from **rpm** (only binaries) all these dependencies have been removed except *mysqlclient*. However, they are needed when running EAR.
- SLURM must also be present if the SLURM plugin wants to be used. Since current EAR version only supports automatic execution of applications with EAR library using the SLURM plugin, it must be running when EAR library wants to be used (not needed for node monitoring).
- The drivers for CPUFreq management (*acpi-cpufreq*) and Open IPMI must be present and loaded in compute nodes.
- *msr kernel* module must be loaded in compute nodes.
- mariaDB or postgres server must be up and running.
- Hardware counters must be accessible for normal users. Set */proc/sys/kernel/perf_event_paranoid* to 2 (or less). Type `sudo sh -c "echo 2 > /proc/sys/kernel/perf_event_paranoid"` in compute nodes.

Installation, configuration and execution

1. Compile and install from source code or install via rpm. `$EAR_TMP` and `$EAR_ETC` are defined in ear module. Till the module is not loaded, define manually these environment variables to execute the next steps.
2. Create the `$EAR_TMP` folder. This folder must be local to each node, so we recommend to create it in `/var/ear`.
3. Either installing from sources or rpm, EAR installs a template for **ear.conf** file in `$EAR_ETC/ear/ear.conf.template`. Copy at `$EAR_ETC/ear/ear.conf` and update with the desired configuration. Go to ear.conf page to see how to do it. The ear.conf is used by all the services.
4. Load EAR module to enable commands. It can be found in `$EAR_ETC/module`. You can add ear module when it's not in standard paths by doing `module use $EAR_ETC/module` and then `module load ear`.
5. Create EAR database with `edb_create`. The `edb_create -p` command will ask you for the DB root password. If you get any problem here, check first whether the node where you are running the command can connect to the DB server. In case problems persists, execute `edb_create -o` to report the specific SQL queries generated. In case of trouble, contact with ear-support@bsc.es.
6. EAR uses a power and performance model based on systems signatures. These system signatures are stored in coefficient files. Before starting EARD, and just for testing, it is needed to create a dummy coefficient file and copy in the coefficients path (by default placed at `$EAR_ETC/coeffs`). Visit the *coeffs_null* application from tools section.
7. Copy EAR service files to start/stop services using system commands such as `systemctl`. EAR service files are generated at `$EAR_ETC/systemd` and they can usually be placed in `$(ETC)/systemd`.
8. Start EARDs and EARDBDs via services (see our Launching the components with unit services). EARDBD and EARD outputs can be found at `$EAR_TMP/eardbd.log` and `$EAR_TMP/eard.log` respectively when *DBDaemonUseLog* and *NodeUseLog* options are set to 1 in ear.conf file. Otherwise, their outputs are generated in *stderr* and can be seen using the *journactl* command. For instance, use `journactl -u eard` to look at eard output.
9. Check that EARDs are up and running correctly with `econtrol --status` (note that daemons will take around a minute to correctly report energy and not show up as an error in `econtrol`). EARDs create a per-node text file with values reported to the EARDBD. In case there are problems when running *econtrol*, you can also find this file at `$EAR_TMP/nodename.pm_periodic_data.txt`.
10. Check that EARDs are reporting metrics to database with *ereport*. `ereport -n all` should report the total energy send by each daemon since the setup.

11. Start EARGM via services.
12. Check if EARGM is reporting to database with `ereport -g`. Note that EARGM will take a period of time set by the admin in `ear.conf` (`GlobalManagerPeriodT1` option) to report for the first time.
13. Set up EAR's SLURM plugin (see our Configuration page for more information).
14. Run an application via SLURM and check that it is correctly reported to database with `eacct`. Note that only privileged users can check other users' applications.
15. Run an MPI application with `--ear=on` and check that the report by `eacct` now includes the library metrics. EAR library depends on the MPI version: Intel, OpenMPI, etc. By default `libear.so` is used. Different names for different versions can be specified automatically by adding the EAR version name in the corresponding MPI module. For instance, for `libear.openmpi.4.0.0.so` library, define `SLURM_EAR_MPI_VERSION` environment variable as `openmpi.4.0.0`. When EAR has been installed from sources, this name is the same as it is specified in `MPI_VERSION` during the `configure`. When installed from rpm, look at `$EAR_INSTALL_PATH/lib` to see the available versions.
16. Set `default=on` to specify the EAR library will be loaded with all the applications by default in `plugstack.conf`. If default is set to off, EAR library can be explicitly loaded by doing `--ear=on` when submitting a job.
17. At this point you can use EAR for monitoring and accounting purposes, but it cannot use the power policies for EARL. To do that, first do a learning phase and compute the coefficients.
18. For the coefficients to be active, restart the daemons. **IMPORTANT**: reloading the daemons will NOT make them load the coefficients, restarting is the only way.

Requirements

EAR requires some third party libraries 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**, minimum **tested** versions and its references:

Library	Minimum version	References
SLURM	17.02.6	Website (https://slurm.schedmd.com/)
MPI	-	-
MySQL*	15.1	MySQL (https://mysql.com) or MariaDB (https://mariadb.org/)
PostgreSQL*	9.2	PostgreSQL (https://www.postgresql.org/)
CUDA**	7.5	CUDA (https://mysql.com) or MariaDB (https://mariadb.org/)
Autoconf	2.69	Website (https://www.gnu.org/software/autoconf/autoconf.html)
GSL	1.4	Website (https://www.gnu.org/software/gsl/)

* Just one of them required.

** Required if you want to monitor GPU data.

Also, some **drivers** has to be present and loaded in the system:

Driver	File	Kernel version	References
CPUFreq	kernel/drivers/cpufreq/acpi-cpufreq.ko	3.10	Information (https://wiki.archlinux.org/index.php/CPU_frequency_scaling)
Open IPMI	kernel/drivers/char/ipmi/*.ko	3.10	Information (https://docs.oracle.com/en/database/oracle/oracle-database/12.2/cwlin/configuring-the-open-ipmi-driver.html)

Lastly, the **compilers**: EAR uses C compilers. It has been tested with both Intel and GNU.

Compiler	Comment	Minimum version	References
GNU Compiler Collection (GCC)	For the library and daemon	4.8.5	Website (https://gcc.gnu.org/)
Intel C Compiler (ICC)	For the library and daemon	17.0.1	Website (https://software.intel.com/en-us/c-compilers)

Compilation and installation guide summary

1. Before the installation, make sure the installation path is accessible by all the computing nodes. Do the same in the folder where you want to set the configuration files (it will be called `$(EAR_ETC)` in this guide for simplicity).
2. Generate Autoconf's `configure` program by typing `autoreconf -i`.
3. Read sections below to understand how to properly set the `configure` parameters.
4. Compile EAR components by typing `./configure ...`, `make` and `make install` in the root directory.
5. Type `make etc.install` to install the content of `$(EAR_ETC)`. It is the configuration content, but that configuration will be expanded in the next section. You have a link at the bottom of this page.

Configure options

`configure` is based on shell variables which initial value could be given by setting variables in the command line, or in the environment. Take a look to the table with the most popular variables:

Variable	Description
MPICC	MPI compiler.
CC	C compiler command.
MPICC_FLAGS	MPI compiler flags.
CFLAGS	C compiler flags.
CC_FLAGS	Also C compiler flags.
LDFLAGS	Linker flags. E.g. '-L' if you have libraries in a nonstandard directory \.
LIBS	Libraries to pass to the linker. E.g. '-l'.
EAR_TMP	Defines the node local storage as 'var', 'tmp' or other tempfs file system (default: /var/ear) (you can also use <code>--localstatedir=DIR</code>).
EAR_ETC	Defines the read-only single-machine data as 'etc' (default: EPREFIX/etc) (you can also use <code>--sharedstatedir=DIR</code>).
MAN	Defines the manual directory (default: PREFIX/man) (you can use also <code>--mandir=DIR</code>).
DOC	Defines the documentation directory (default: PREFIX/doc) (you can use also <code>--docdir=DIR</code>).
MPI_VERSION	Adds a suffix to the compiled EAR library name. Read further down this page for more information.
USER	Owner user of the installed files.
GROUP	Owned group of the installed files

- This is an example of `CC`, `CFLAGS` and `DEBUG` variables overwriting:

```
./configure CC=icc CFLAGS=-g EAR_ETC=/hpc/opt/etc
```

You can choose the root folder by typing `./configure --PREFIX=<path>`. But there are other options in the following table:

Definition	Default directory	Content / description
\<PREFIX>	/usr/local	Installation path
\<EAR_ETC>	\<PREFIX>/etc	Configuration files.
\<EAR_TMP>	/var/ear	Pipes and temporal files.

You have more installation options information by typing `./configure --help`. If you want to change the value of any of this options after the configuration process, you can edit the root Makefile. All the options are at the top of the text and its names are self-explanatory.

ADDING REQUIRED LIBRARIES INSTALLED IN CUSTOM LOCATIONS

The `configure` script is capable to find libraries located in custom location if a module is loaded in the environment or its path is included in `LD_LIBRARY_PATH`. If not, you can help `configure` to find SLURM, or other required libraries in case you installed in a custom location. It is necessary to add its root path for the compiler to see include headers and libraries for the linker. You can do this by adding to it the following arguments:

Argument	Description
<code>--with-slurm=\<path></code>	Specifies the path to SLURM installation.
<code>--with-cuda=\<path></code>	Specifies the path to CUDA installation.
<code>--with-mysql=\<path></code>	Specify path to MySQL installation.
<code>--with-pgsql=\<path></code>	Specify path to PostgreSQL installation.
<code>--with-gsl=\<path></code>	Specifies the path to GSL installation.
<ul style="list-style-type: none"> This is an example of <code>cc</code> overwriting the CUDA path specification: <code>./configure --with-cuda=/path/to/CUDA</code> 	

If unusual procedures must be done to compile the package, please try to figure out how `configure` could check whether to do them and contact the team to be considered for the next release. In the meantime, you can overwrite shell variables or export its paths to the environment (e.g. `LD_LIBRARY`).

ADDITIONAL CONFIGURE FLAGS

Also, there are additional flags to help administrator increase the compatibility of EAR in nodes.

Argument	Description
<code>--disable-rpath</code>	Disables the RPATH included in binaries to specify some dependencies location.
<code>--disable-avx512</code>	Replaces the AVX-512 function calls by AVX-2.
<code>--disable-gpus</code>	The GPU monitoring data is not allocated nor inserted in the database.

Pre-installation fast tweaks

Some EAR characteristics can be modified by changing the value of the constants defined in `src/common/config/config_def.h`. You can open it with an editor and modify those pre-processor variables to alter the EAR behaviour.

Also, you can quickly switch the user/group of your installation files by modifying the `CHOWN_USR/CHOWN_GRP` variables in the root Makefile.

Library distributions/versions

As commented in the overview, the EAR library is loaded next to the user MPI application by the EAR Loader. The library uses MPI symbols, so it is compiled by using the includes provided by your MPI distribution. The selection of the library version is automatic in runtime, but in the compiling and installation process is not required. Each compiled library has its own file name that has to be defined by the `MPI_VERSION` variable during `./configure` or by editing the root Makefile. The name list per distribution is exposed in the following table:

DistributionName	MPI_VERSION variable
Intel MPI	libear.so (default)it is not required
MVAPICH	libear.so (default)it is not required
OpenMPI	libear.ompi.so ompi

If different MPI distributions shares the same library name, it means that its symbols are compatible between them, so compiling and installing the library one time will be enough. However, if you provide different MPI distributions to the users, you will have to compile and install the library multiple times.

Before compiling new libraries you have to install by typing `make install`. Then you can run the `./configure` again, changing the `MPICC`, `MPICC_FLAGS` and `MPI_VERSION` variables, or just opening the root Makefile and edit the same variables and `MPI_BASE`, which just sets the MPI installation root path. Now type `make full` to perform a clean compilation and `make ear1.install`, to install only the new version of the library.

If your MPI version is not fully compatible, please contact ear-support@bsc.es. We will add compatibility to EAR and give you a solution in the meantime.

Other useful flags

You can install individual components by doing: `make eard.install` to install EAR Daemon, `make ear1.install` to install EAR Library, `make eardbd.install` EAR Database Manager, `make eargmd.install` EAR Global Manager and `make commands.install` the EAR command binaries.

Installation content

This is the list of the inner installation folders and their content:

Root	Directory	Content / description
\<PREFIX>	/lib	Libraries.
\<PREFIX>	/lib/plugins	Plugins.
\<PREFIX>	/bin	EAR commands.
\<PREFIX>	/bin/tools	EAR tools for coefficients.
\<PREFIX>	/sbin	Privileged components.
\<PREFIX>	/man	Documentation.
\<EAR_ETC>	/ear	Configuration file.
\<EAR_ETC>	/ear/coeffs	Coefficient files store.
\<EAR_ETC>	/module	EAR module.
\<EAR_ETC>	/slurm	ear.pluginstack.conf.
\<EAR_ETC>	/systemd	EAR service files.

Fine grain tuning of EAR options

Some options such as the maximum number of CPUs or GPUs supported are defined in `src/common/config` files. It is not recommended to modify these files but some options and default values can be set by modifying them.

Next step

For a better overview of the installation process, return to our Quick installation guide. To continue the installation, visit the configuration page to set up properly the EAR configuration file and the SLURMs plugin stack file.

Requirements

EAR uses some third party libraries. EAR RPM will not ask for them when installing but they must be available in `LD_LIBRARY_PATH` when running an application and you want to use EAR. Depending on the RPM, different version must be required of this libraries:

Library	Minimum version	References
SLURM	17.02.6	Website (https://slurm.schedmd.com/)
MPI	-	-
MySQL*	15.1	MySQL (https://mysql.com) or MariaDB (https://mariadb.org/)
PostgreSQL*	9.2	PostgreSQL (https://www.postgresql.org/)
Autoconf	2.69	Website (https://www.gnu.org/software/autoconf/autoconf.html)
GSL	1.4	Website (https://www.gnu.org/software/gsl/)

(*) Just one of them required.

Also, some **drivers** has to be present and loaded in the system when starting EAR:

Driver	File	Kernel version	References
CPUFreq	<code>kernel/drivers/cpufreq/acpi-cpufreq.ko</code>	3.10	Information (https://wiki.archlinux.org/index.php/CPU_frequency_scaling)
Open IPMI	<code>kernel/drivers/char/ipmi/*.ko</code>	3.10	Information (https://docs.oracle.com/en/database/oracle/oracle-database/12.2/cwlin/configuring-the-open-ipmi-driver.html)

Installation guide

1. Before the installation, make sure the installation path is accessible by all the computing nodes. Do the same in the folder where you want to set the temporary files (it will be called `$(EAR_TMP)` in this guide for simplicity).
2. Default paths are `/usr` and `/etc`
3. Run `rpm -ivh --relocate /usr=/new_install_path --relocate /etc=/new_etc_path ear.version.rpm`. You can also use the `--nodeps` if your dependency test fails.
4. During the installation the configuration files `*.in` are compiled to the ready to use version, replacing tags

for correct paths. You will have more information of those files in the following pages. Check the next section for more information

5. To uninstall the RPM type `rpm -e ear.version`.

Installation content

Directory	Content / description
/usr/lib	Libraries
/usr/lib/plugins	Plugins
/usr/bin	EAR commands
/usr/bin/tools	EAR tools for coefficients
/usr/sbin	Privileged components: EARD, EARDBD, EARGMD
/etc/ear	Configuration file templates
/etc/ear/coeffs	Folder to store coefficient files.
/etc/module	EAR module.
/etc/slurm	ear.pluginstack.conf
/etc/systemd	EAR service files

The *.in configuration files are compiled into `etc/ear/ear.conf.template` and `etc/ear/ear.full.conf.template`, `etc/module/ear`, `etc/slurm/ear.pluginstack.conf` and various `etc/systemd/ear*.service`. You can find more information in the next configuration section.

Next step

For a better overview of the installation process, return to our Quick installation guide. To continue the installation, visit the configuration page to set up properly the EAR configuration file and the SLURMs plugin stack file.

Configuration requirements

The following requirements must be met for EAR to work properly:

- EAR folders: EAR uses two paths for EAR configuration.
 - **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. It 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 has very few data and it is 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. Both **ear.conf** and coefficients must be readable in all the nodes (compute and "service" nodes).
- 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. For more details, check EAR configuration file below.
- MySQL DB or PostgreSQL DB: EAR saves data in a MySQL/PostgreSQL DB server. EAR DB can be created using `edb_create` command provided (MySQL/PostgreSQL server must be running and root access to the DB is needed).
- Set EAR SLURM plugin
 - EAR SLURM plugin must be set in `/etc/slurm/plugstack.conf`. EAR generates an example at *ear_etc_path/slurm/ear.plugstack.conf*. For more information see our Plugin section down below.

EAR configuration file

ear.conf is a text file describing the EAR package behaviour in the cluster. It must be readable by all compute nodes and by nodes where commands are executed. Two **ear.conf** templates are generated with default values and will be installed as reference when executing `make etc.install`

Usually the first word in the configuration file expresses the component related with the option. Lines starting with `#` are comments.

A test for **ear.conf** file can be found in the path `src/test/functionals/ear_conf`.

In-depth EAR configuration file options

Database configuration

```

# The IP of the node where the MariaDB (MySQL) or PostgreSQL server process is running. Current
DBIp=172.30.2.101
# Port in which the server accepts the connections.
DBPort=3306
# MariaDB user that the services will use. Needs INSERT/SELECT privileges. Used by EARDBD
DBUser=eardbd_user
# Password for the previous user. If left blank or commented it will assume the user has no pas
DBPassw=eardbd_pass
# Database user that the commands (eacct, ereport) will use. Only uses SELECT privileges.
DBCommandsUser=ear_commands
# Password for the previous user. If left blank or commented it will assume the user has no pas
DBCommandsPassw=commandspass
# Name of EAR's database in the server.
DBDatabase=EAR
# Maximum number of connections of the commands user to prevent server saturation/malicious act
DBMaxConnections=20
# The following specify the granularity of data reported to database.
# Extended node information reported to database (added: temperature, avg_freq, DRAM and PCK en
DBReportNodeDetail=1
# Extended signature hardware counters reported to database.
DBReportSigDetail=1
# Set to 1 if you want Loop signatures to be reported to database.
DBReportLoops=1

```

EARD configuration. EARD are executed in compute nodes

```

# The port where the EARD will be listening.
NodeDaemonPort=50001
# Frequency used by power monitoring service, in seconds.
NodeDaemonPowermonFreq=60
# Maximum supported frequency (1 means nominal, no turbo).
NodeDaemonMaxPstate=1
# Enable (1) or disable (0) the turbo frequency.
NodeDaemonTurbo=0
# Enables the use of the database.
NodeUseDB=1
# Inserts data to MySQL by sending that data to the EARDBD (1) or directly (0).
NodeUseEARDBD=1
# '1' means EAR is controlling frequencies at all times (targeted to production systems) and 0
NodeDaemonForceFrequencies=1
# The verbosity level [0..4]
NodeDaemonVerbose=1
# When set to 1, the output is saved in '$EAR_TMP'/eard.log (common configuration) as a log fil
NodeUseLog=1
# Minimum time between two energy readings for performance accuracy
MinTimePerformanceAccuracy=1000000

```

EARDBD configuration

```
# Port where the EARDBD server is listening
DBDaemonPortTCP=50002
# Port where the EARDBD mirror is listening
DBDaemonPortSecTCP=50003
# Port is used to synchronize the server and mirror
DBDaemonSyncPort=50004
# In seconds, interval of time of accumulating data to generate an energy aggregation
DBDaemonAggregationTime=60
# In seconds, time between inserts of the buffered data
DBDaemonInsertionTime=30
# Memory allocated per process. This allocations is used for buffering the data sent to the dat
DBDaemonMemorySize=120
# When set to 1, eardbd uses a '$EAR_TMP'/eardbd.log file as a log file
DBDaemonUseLog=1
```

EARL configuration

```
# Path where coefficients are installed, usually $EAR_ETC/ear/coeffs
CoefficientsDir=/path/to/coeffs
# Number of levels used by DynAIS algorithm.
DynAISLevels=10
# Windows size used by DynAIS, the higher the size the higher the overhead.
DynAISWindowSize=200
# Maximum time in seconds that EAR will wait until a signature is computed. After this value, i
DynaisTimeout=15
# Time in seconds to compute every application signature when the EAR goes to periodic mode.
LibraryPeriod=10
# Number of MPI calls whether EAR must go to periodic mode or not.
CheckEARModeEvery=1000
```

EARGM configuration

```

# The IP or hostname of the node where the EARGMD daemon is running.
EARGMHost=hostname
# Port where EARGMD will be listening.
EARGMPort=50000
# Use '1' or not '0' aggregated metrics to compute total energy.
EARGMUseAggregated=1
# Period T1 and period T2 are specified in seconds. T1 must be less than T2. Global manager upd
EARGMPeriodT1=90
EARGMPeriodT2=259200
# Units field, Can be '-' (Joules), 'K' KiloJoules or 'M' MegaJoules
EARGMUnits=K
# This limit means the maximum energy allowed in 259200 seconds in 550000 KJoules
EARGMEnergyLimit=550000
#
# Global manager modes. Two modes are supported '0' (manual) or '1' (automatic). Manual means G
EARGMMode=0
# A mail can be sent reporting the warning level (and the action taken in automatic mode). 'nom
EARGMMail=nomail
# Percentage of accumulated energy to start the warning DEFCON level L4, L3 and L2.
EARGMWarningsPerc=85,90,95
# Number of "grace" T1 periods before doing a new re-evaluation. After a warning, EARGM will wa
EARGMGracePeriods=3
# Verbose level
EARGMVerbose=1
# When set to 1, the output is saved in '$EAR_TMP'/eargmd.log (common configuration) as a log f
EARGMUseLog=1
# Format for action is: command_name energy_T1 energy_T2 energy_limit T2 T1 units "
# This action is automatically executed at each warning level (only once per grace periods)
EARGMEnergyAction=no_action
#### POWERCAP definition for EARGM: Powercap is still under development. Do not activate
# 0 means no powercap
EARGMPowerLimit=0

```

Common configuration

```
# Default verbose level
Verbose=0
# Path used for communication files, shared memory, etc. It must be PRIVATE per compute node an
TmpDir=/tmp/ear
# Path where coefficients and configuration are stored. It must be readable in all compute node
EtcDir=/path/to/etc
InstDir=/path/to/inst
# Path where metrics are generated in text files when no database is installed. A suffix is inc
DataBasePathName=/etc/ear/dbs/dbs.
# Energy reading plugin (without the extension). Allows to use different system components to r
# look at /path/to/inst/lib/plugins/energy folder to see the list of installed energy plugins
Energy_plugin=energy_nm.so
# Power model plugin (without the extension). The power model plugin is used to predict the pow
Energy_model=avx512_model.so
```

EAR-Authorized users/groups/accounts

Authorized users that are allowed to change policies, thresholds and frequencies are supposed to be administrators. A list of users, Linux groups, and/or SLURM accounts can be provided to allow normal users to perform that actions. Only normal Authorized users can execute the learning phase.

```
AuthorizedUsers=user1,user2
AuthorizedAccounts=acc1,acc2,acc3
AuthorizedGroups=xx,yy
```

Energy tags

Energy tags are pre-defined configurations for some applications (EAR library is not loaded). This energy tags accept a user ids, groups and SLURM accounts of users allowed to use that tag.

```
# General energy tag
EnergyTag=cpu-intensive pstate=1
# Energy tag with limited users
EnergyTag=memory-intensive pstate=4 users=user1,user2 groups=group1,group2 accounts=acc1,acc2
```

Tags

Tags are used for architectural descriptions. Max. AVX frequencies are used in predictor models and are SKU-specific. At least a default tag is mandatory to be included for a cluster to work properly.

The `min_power`, `max_power` and `error_power` are threshold values that determine if the metrics read might be invalid, and a warning message to syslog will be reported if the values are outside of said thresholds.

`error_power` is a more extreme value that if a metric surpasses it, said metric will not be reported to database.

A special energy plugin or energy model can be specified in a tag that will override the global values

previously defined in all nodes that have this tag associated with them.

```
Tag=6148 default=yes max_avx512=2.2 max_avx2=2.6 max_power=500 min_power=50 error_power=600 coe
Tag=6126 max_avx512=2.3 max_avx2=2.9 ceffs=coeffs.6126.default max_power=600 error_power=700
```

Power policies plugins

```
#-----
## Power policies
## -----
#
## policy names must be exactly file names for policies installed in the system
DefaultPowerPolicy=monitoring
Policy=monitoring Settings=0 DefaultFreq=2.4 Privileged=0
Policy=min_time Settings=0.7 DefaultFreq=2.0 Privileged=0
Policy=min_energy Settings=0.05 DefaultFreq=2.4 Privileged=1

# For homogeneous systems, default frequencies can be easily specified using freqs, for heterog

# Example with pstates (lower pstates corresponds with higher frequencies). Pstate=1 is nominal
#Policy=monitoring Settings=0 DefaultPstate=1 Privileged=0
#Policy=min_time Settings=0.7 DefaultPstate=4 Privileged=0
#Policy=min_energy Settings=0.05 DefaultPstate=1 Privileged=1
```

Island description

This section is mandatory since it is used for cluster description. Normally nodes are grouped in islands that share the same hardware characteristics as well as its database managers (EARDBDS). Each entry describes part of an island, and every node must be in an island.

There are two kinds of database daemons. One called 'server' and other one called 'mirror'. Both performs the metrics buffering process, but just one performs the insert. The mirror will do that insert in case the 'server' process crashes or the node fails.

It is recommended for all islands to have maintain server-mirror symmetry. For example, if the island I0 and I1 have the server N0 and the mirror N1, the next island would have to point the same N0 and N1 or point to new ones N2 and N3, not point to N1 as server and N0 as mirror.

Multiple EARDBDs are supported in the same island, so more than one line per island is required, but the condition of symmetry have to be met.

It is recommended that for an island the server and the mirror to be running in different nodes. However, the EARDBD program could be both server and mirror at the same time. This means that the islands I0 and I1 could have the N0 server and the N2 mirror, and the islands I2 and I3 the N2 server and N0 mirror, fulfilling the symmetry requirements.

A tag can be specified that will apply to all the nodes in that line. If no tag is defined, the default one will be used as hardware definition.

```
#
# In the following example the nodes are clustered in two different islands, but the Island 1 h
# two types of EARDBDs configurations.
#

Island=0 DBIP=node1081 DBSECIP=node1082 Nodes=node10[01-80]

# These nodes are in island0 using different DB connections and with a different architecture
Island=0 DBIP=node1084 DBSECIP=node1085 Nodes=node11[01-80] DBSECIP=node1085 tag=6126
# These nodes are in island0 and will use default values for DB connection (line 0 for island0)
Island=0 Nodes=node12[01-80]

# Will use default tag
Island=1 DBIP=node1181 DBSECIP=node1182 Nodes=node11[01-80]
```

Detailed island accepted values:

- `nodename_list` accepts the following formats:
 - `Nodes= node1,node2,node3`
 - `Nodes= node[1-3]`
 - `Nodes= node[1,2,3]`
- Any combination of the two latter options will work, but if nodes have to be specified individually (the first format) as of now they have to be specified in their own line. As an example:
 - Valid formats:
 - `Island=1 Nodes= node1,node2,node3`
 - `Island=1 Nodes= node[1-3],node[4,5]`
 - Invalid formats:
 - `Island=1 Nodes= node[1,2],node3`
 - `Island=1 Nodes= node[1-3],node4`

SLURM spank plugin configuration file

SLURM loads the plugin through a file called `plugstack.conf`, which is composed by a list of a plugins. In the file `etc/slurm/ear.plugstack.conf`, there is an example entry with the paths already set to the plugin, temporal and configuration paths.

Example:

```
required ear_install_path/lib/earplug.so prefix=ear_install_path sysconfdir=etc_ear_path local
```

The argument `prefix` points to the EAR installation path and it is used to load the library using `LD_PRELOAD` mechanism. Also the `localstatedir` is used to contact with the EARD, which by default points the path you set during the `./configure` using `--localstatedir` or `EAR_TMP` arguments. Next to these fields, there is the field `earlib_default=off`, which means that by default EARL is not loaded. Finally there are `eargmd_host` and `eargmd_port` if you plan to connect with the EARGMD component (you can leave this empty).

Also, there are two additional arguments. The first one, `nodes_allowed=` followed by a comma separated list

of nodes, enables the plugin only in that nodes. The second, `nodes_excluded=` , also followed by a comma separated list of nodes, disables the plugin only in nodes in the list. These are arguments for very specific configurations that must be used with caution, if they are not used it is better that they are not written.

Example:

```
required ear_install_path/lib/earplug.so prefix=ear_install_path sysconfdir=etc_ear_path local
```

MySQL/PostgreSQL

WARNING: If any EAR component is running in the same machine as the MySQL server some connection problems might occur. This will not happen with PostgreSQL. To solve those issues, input into MySQL's CLI client the `CREATE USER` and `GRANT PRIVILEGES` queries from `edb_create -o` changing the portion `'user_name'@'%'` to `'user_name'@'localhost'` so that EAR's users have access to the server from the local machine. There are two ways to configure a database server for EAR's usage.

- run `edb_create -r` located in `$EAR_INSTALLATION_PATH/sbin` from a node with root access to the MySQL server. This requires MySQL/PostgreSQL's section of `ear.conf` to be correctly written. For more info run `edb_create -h` .
- Manually create the database and users specified in `ear.conf`, as well as the required tables. If `ear.conf` has been configured, running `edb_create -o` will output the queries that would be run with the program that contain all that is needed for EAR to properly function.

For more information about how each `ear.conf` flag changes the database creation, see our Database section.

Next step

Visit the execution page to run EAR's different components.

Tools list

Name	Description	Basic arguments
<code>coeffs_compute</code>	Computes the learning coefficients	<code><save.path></code> <code><min.frequency></code> <code><node.name></code>
<code>coeffs_default</code>	Computes a default coefficients file	
<code>coeffs_null</code>	Created a dummy configuration file to be used by EARD	<code>coeff_path</code> , <code>max.freq</code> <code>min.freq</code>
<code>coeffs_show</code>	Shows the computed coefficients file in text format	<code><file.path></code>

- Use the argument `--help` to expand the application information and list the admitted flags.

Examples

- Compute the coefficients for the node `node1001` in which the minimum frequency set during the learning phase was 1900000 KHz
- ```
./coeffs_compute /etc/coeffs 1900000 node1001
```

This is a necessary phase prior to the normal EAR utilization and is a kind of hardware characterization of the nodes. During the phase a matrix of coefficients are calculated and stored. These coefficients will be used to predict the energy consumption and performance of each application.

Please, visit the learning phase wiki page ([https://gitlab.bsc.es/ear\\_team/ear\\_learning/-/wikis/home](https://gitlab.bsc.es/ear_team/ear_learning/-/wikis/home)) to read the manual and the repository ([https://gitlab.bsc.es/ear\\_team/ear\\_learning](https://gitlab.bsc.es/ear_team/ear_learning)) to get the scripts and the kernels.

## EAR plug-ins

---

Some of the core of EAR functionality can be dynamically loaded through a plug-in mechanism, making EAR more extensible and dynamic than previous version since it is not needed to reinstall the system to add , for instance, a new policy or a new power model. It is only needed to copy the file in the `$EAR_INSTALL_PATH/lib/plugin` folder and restart some components. The three parts that can be loaded as plug-ins are: the **node energy reading** library, the **power policy**, the **power model** and the **tracing**.

| Plug-in         | Description                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------------------|
| Power model     | It is used to predict the energy consumption given a target frequency and the current state metrics.      |
| Power policies  | Defines the behaviour of EAR to switch between frequencies given energy readings and predictions.         |
| Energy readings | It is used to read the energy of the node.                                                                |
| Tracing         | EAR library data and internal states changes are exported to the tracing library in case it is specified. |

## Considerations

---

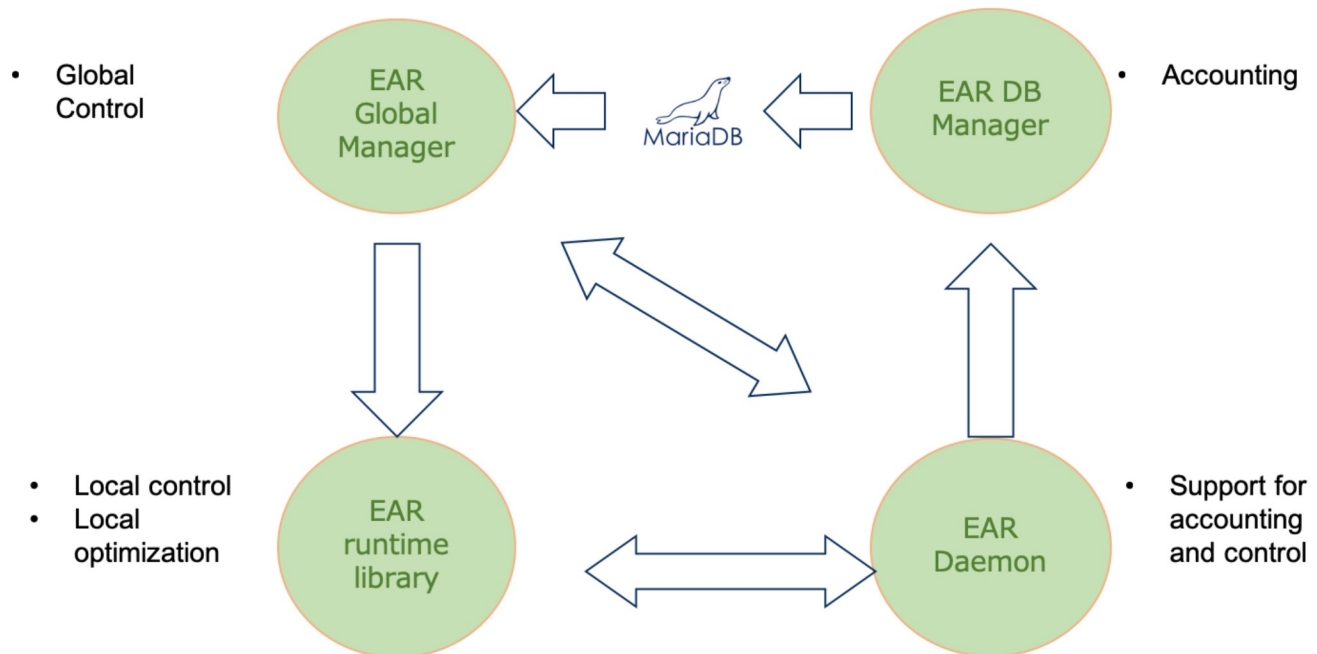
- Plug-in **paths** is set by default to `$EAR_INSTALL_PATH/lib/plugin`.
- Default **power model** library is specified in `ear.conf` (*energy\_model* option). By default EAR includes a `basic_model.so` and `avx512_model.so` plug-ins.
- The **node energy readings** library is specified in `ear.conf` in the *energy\_plugin* option. Three libraries are included: `energy_nm.so` (uses Intel NodeManager IPMI commands), `energy_rapl.so` (uses a node energy estimation based on DRAM and PACKAGE energy provided by RAPL), and `energy_sd650.so` (uses the high frequency IPMI hardware included in Lenovo SD650 systems).
- **Power policies** included in EAR are: `monitoring.so`, `min_energy.so`, `min_time.so`, `min_energy_no_models.so` and `min_time_no_models.so`. The list of policies installed is automatically detected by the EAR plug-in. However, only policies included in `ear.conf` can be used.
- The **tracing** is an optional functionality. It is included to provide additional information or to generate runtime information.

**Note:** SLURM Plugin does not fit in this philosophy, it is a core component of EAR and can not be replaced by any third party development.

EAR is composed of five main components:

- Node Manager (EARD)
- Database Manager (EARDBD)
- Global Manager (EARGM)
- Library (EARL)
- Loader (EARLO)
- SLURM plugin

The following image shows the main interactions between components:



## Node Manager

---

EAR's daemon is a per-node process that provides privileged metrics of each node as well as a periodic power monitoring service. Said periodic power metrics are sent to EAR's database either directly or via the database daemon (see configuration page).

For more information, see EARD.

## Database Manager

---

The database daemon acts as an intermediate layer between any EAR component that inserts data and the EAR's database, in order to prevent the database server from collapsing due to getting overrun with connections and insert queries.

For more information, see EARDBD.

## Global Manager

---

EAR's Global Manager Daemon (EARGMD) is a cluster wide component that controls the percentage of the maximum energy consumed.

For more information, see EARGM.

## Library

---

The EAR library is the core of the EAR package. The EARL offers a lightweight and simple solution to select the optimal frequency for MPI applications at runtime, with multiple power policies each with a different approach to find said frequency. EARL uses the daemon to read performance metrics and to send application data to EAR's database.

For more information, see EARL.

## Loader

---

EAR Loader is the responsible for loading the EAR Library. It is a small and lightweight library loaded by the SLURM Plugin, that identifies the user application and loads its corresponding EAR Library distribution.

For more information, see EARLO.

## SLURM Plugin

---

EAR SLURM plugin allows to dynamically load and configure the EAR library for the SLURM jobs, if the enabling argument is set or is enabled by default. Additionally, it reports any jobs that start or end to the nodes' EARDs for accounting and monitoring purposes.

For more information, see SLURM Plugin.

## EARD: Node Manager

---

The node daemon is the component in charge of providing any kind of services that requires privileged capabilities. Current version is conceived as an external process executed with root privileges.

The EARD provides the following services, each one covered by one thread:

- Provides privileged metrics to EARL such as the average frequency, uncore integrated memory controller counters to compute the memory bandwidth, as well as 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 a remote API used by EARplug, EARGM and EAR commands. This API accepts requests such as get the system status, change policy settings or notify new job/end job events.

## Requirements

---

When executed in production environments, EARD connects with EARDBD service, that has to be up before starting the node daemon, otherwise values reported by EARD to be stored in the database, will be lost.



## Configuration

---

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

Please visit the EAR configuration file page for more information about the options of EARD and other components.

## Execution

---

To execute this component, these `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 reloading 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`

## Reconfiguration

---

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

```
DefaultPstates
NodeDaemonMaxPstate
NodeDaemonVerbose
NodeDaemonPowermonFreq
SupportedPolicies
MinTimePerformanceAccuracy
```

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

## EARDBD: Database Manager

---

EARDBD caches records generated by the EARL and EARD in the system and reports it to the centralized database. It is recommended to run several EARDBDs if the cluster is big enough in order to reduce the number of inserts and connections to the database.

Also, EARDBD accumulates data during a period of time to decrease the total insertions in the database, helping the performance of big queries. By now just the energy metrics are available to accumulate in the new metric called energy aggregation. EARDBD uses periodic power metrics sent by EARD, the per-node daemon, including job identification details (job id and step id when executed in a SLURM system).

## 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.

Please visit the [EAR configuration file page](#) for more information about the options of EARDBD and other components.

## Execution

---

To execute this component, these `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 reloading the configuration of the EARDBD service.

## EARGM: Global Manager

---

EARGM is a cluster wide component offering cluster energy monitoring and capping. EARGM can work in two modes: manual and automatic. When running in manual mode, EARGM monitors the total energy consumption, evaluates the percentage of energy consumption over the energy limit set by the admin and reports the cluster status to the DB. When running in automatic mode, apart from evaluating the energy consumption percentage it sends the evaluation to computing nodes. EARDs passes these messages to EARL which re-applies the energy policy with the new settings.

Apart from sending messages and reporting the energy consumption to the DB, EARGM offers additional features to notify the energy consumption: automatic execution of commands is supported and mails can also automatically be sent. Both the command to be executed or the mail address can be defined in the `ear.conf`, where it can also be specified the energy limits, the monitoring period, etc.

EARGM uses periodic aggregated power metrics to efficiently compute the cluster energy consumption. Aggregated metrics are computed by EARDBD based on power metrics reported by EARD, the per-node daemon.

## Configuration

---

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

Please visit the [EAR configuration file page](#) for more information about the options of EARGM and other components.

## Execution

---

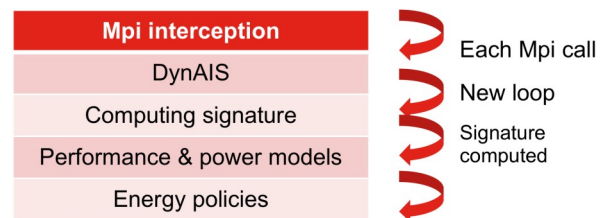
To execute this component, these `systemctl` command examples are provided:

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

## EAR Library

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 next to the running applications by the EAR Loader. Loader intercepts the MPI calls through the PMPI interface, and then calls its respective PMPI function included in the library, which handles the operations. At runtime, EARL goes through the following phases:



1. Automatic **detection** of application outer loops. This is done by intercepting MPI calls and invoking the Dynamic Application Iterative Structure detector algorithm. **DynAIS** is highly optimized for new Intel architectures, reporting low overhead.
2. Computation of the **application signature**. Once DynAIS starts reporting iterations for the outer loop, EAR starts to compute the application signature. This signature includes: iteration time, DC power consumption, bandwidth, cycles, instructions, etc. Since the DC power measurements 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.

$$\text{Power}(fn) = A(Rf,fn) * \text{Power}(Rf) + B(Rf,fn) * \text{TPI}(Rf) + C(Rf,fn)$$

$$\text{CPI}(fn) = D(Rf,fn) * \text{CPI}(Rf) + E(Rf) * \text{TPI}(Rf) + F(Rf,fn)$$

$$\text{TIME}(fn) = \text{TIME}(Rf) * \text{CPI}(Rf,fn) / \text{CPI}(Rf) * (Rf/fn)$$

3. **Power and performance projection**. EAR has its own performance and power models which requires the application and the system signatures as an input. The system signature is a set of coefficients characterizing each node in the system. They are computed during the learning phase at the EAR configuration step. EAR projects the power used and computing time (performance) of the running application for all the available frequencies in the system.





4. **Apply** the selected power policy. EAR includes two power policies to be selected at runtime: *minimize time to solution* and *minimize energy to solution*, if permitted 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 an application and its particular run. An additional policy, *monitoring only* can also be used, but in this case no changes to the running frequency will be made but only the computation and storage of the application signature and metrics will be done.

## Configuration

---

The EAR Library uses the `$(EAR_ETC)/ear.conf` file to be configured. Please visit the EAR configuration file page for more information about the options of EARL and other components.

The library receives its specific settings through a shared memory regions initialized by EARD.

## How to run MPI applications with EARL

---

For information on how to run applications alongside with EARL see our User guides section about it, as well as the Policies page.

## EAR Loader configuration guide

---

Loader is a lightweight and small library loaded by the SLURM Plugin, using the `LD_PRELOAD` environment variable, while executing the user application. Loader detects the underlying application, identifying the MPI version if used and other minor details. With this information, loader opens the suitable EAR Library. As can be read in the EARL page, depending on the MPI vendor, the MPI types can be different, preventing any compatibility between distributions. In example, if the MPI distribution is OpenMPI, the EAR Loader will load the EAR Library compiled with the OpenMPI includes.

## SLURM plugin configuration guide

---

EAR SLURM plugin allows to dynamically load the EAR Loader for the SLURM jobs (and setpid), if the enabling argument is set or if is enabled by default. The Loader will be executed in each job step, intercepting all MPI calls and passing this information to the EAR Library.

## Configuration

---

Visit the configuration page to set up properly the SLURM `/etc/slurm/plugstack.conf` file.

You can find the complete EAR SLURM Plugin parameter in the user guide.

## Tables

---

EAR's database consists of the following tables:

- **Jobs**: job information (app\_id, user\_id, job\_id, step\_id, etc). One record per JOBID.STEPID is created in the DB.
- **Applications**: this table's records serve as a link between Jobs and Signatures, providing an application signature (from EARL) for each node of a job. One record per JOBID.STEPID.NODENAME is created in the DB.
- **Signatures**: EARL computed signature and metrics. One record per JOBID.STEPID.NODENAME is created in the DB when the application is executed with EARL.
- **Periodic\_metrics**: node metrics reported every N seconds (N is defined in `ear.conf`).
- **Periodic\_aggregations**: sum of all *Periodic\_metrics* in a time period to ease accounting in `ereport` command and EARGM, as well as reducing database size (*Periodic\_metrics* of older periods where precision at node level is not needed can be deleted and the aggregations can be used instead).
- **Loops**: similar to *Applications*, but stores a Signature for each application loop detected by EARL, instead of one per each application. This table provides internal details of running applications and could significantly increase the DB size.
- **Events**: EARL events report. Events includes frequency changes, and internal EARL decisions such as turning off the DynAIS algorithm.
- **Global\_energy**: contains reports of cluster-wide energy accounting set by EARGM using the parameters in `ear.conf`. One record every T1 period (defined at `ear.conf`) is reported.
- **Power\_signatures**: Basic time and power metrics that can be obtained without EARL. Reported for all applications. One record per JOBID.STEPID.NODENAME is created in the DB.
- **Learning\_applications**: same as *Applications*, restricted to learning phase applications.
- **Learning\_jobs**: same as *Jobs*, restricted to learning phase jobs.
- **Learning\_signatures**: same as *Signatures*, restricted to learning phase job metrics.

If GPUs are enabled at database creation (or are added afterwards, see Updating from previous versions), *Periodic\_metrics* will also contain GPU data and a new table **GPU\_signatures** will be created, containing all GPU metrics for every application that runs with EARL.

## Database creation and `ear.conf`

---

When running `edb_create` some tables might not be created, or may have some quirks, depending on some `ear.conf` settings. The settings and alterations are as follows:

- `DBReportNodeDetail`: if set to 1, `edb_create` will create two additional columns in the *Periodic\_metrics* table for Temperature (in Celsius) and Frequency (in Hz) accounting.
- `DBReportSigDetail`: if set to 1, *Signatures* will have additional fields for cycles, instructions, and FLOPS1-8 counters (number of instruction by type).
- `DBMaxConnections`: this will restrict the number of maximum simultaneous commands connections.

If any of the settings is set to 0, the table will have fewer details but the table's records will be smaller in stored size.

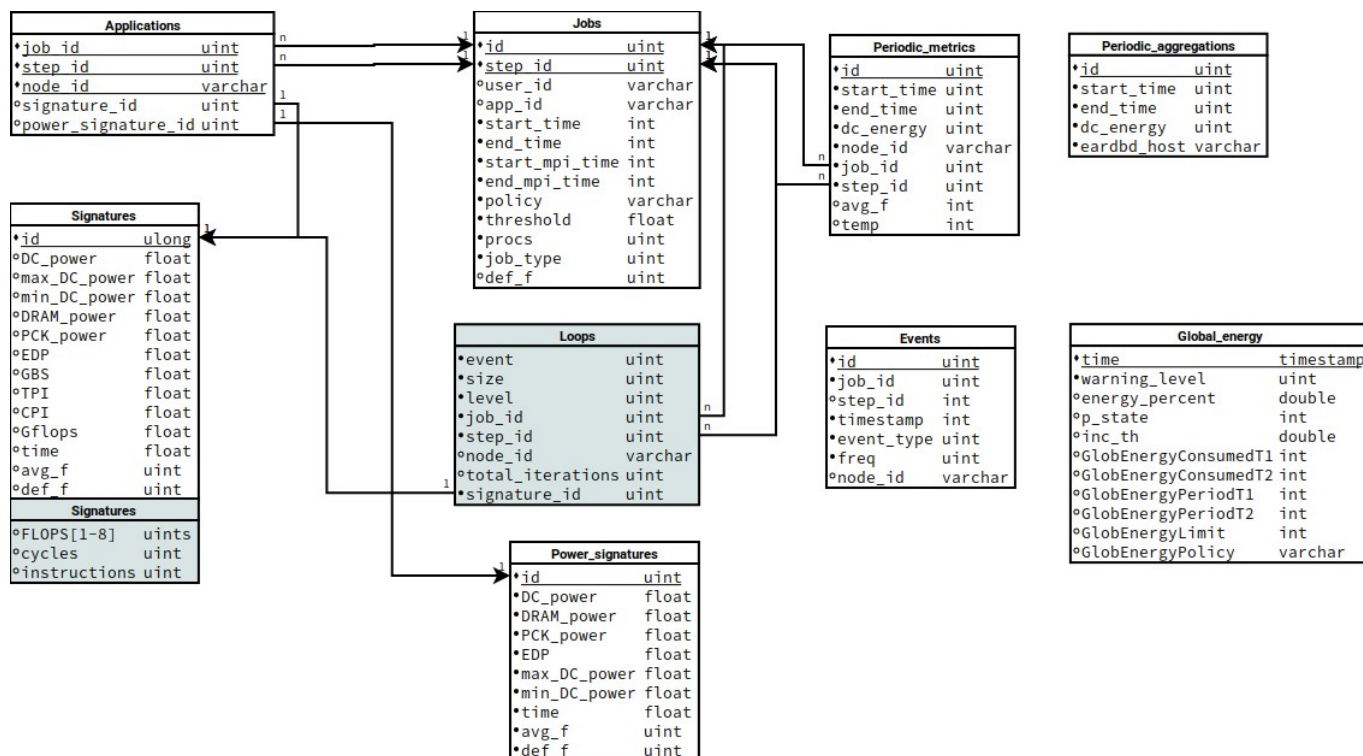
Any table with missing columns can be later altered by the admin to include said columns. For a full detail of each table's columns, run `edb_create -o` with the desired `ear.conf` settings.

## Information reported and `ear.conf`

There are various settings in `ear.conf` that restrict data reported to the database and some errors might occur if the database configuration is different from EARDDB's.

- `DBReportNodeDetail` : if set to 1, node managers will report temperature, average frequency, DRAM and PCK energy to the database manager, which will try to insert it to *Periodic\_metrics*. If *Periodic\_metrics* does not have the columns for both metrics, an error will occur and nothing will be inserted. To solve the error, set `ReportNodeDetail` to 0 or manually update *Periodic\_metrics* in order to have the necessary columns.
- `DBReportSigDetail` : similarly to `ReportNodeDetail` , an error will occur if the configuration differs from the one used when creating the database.
- `DBReportLoops` : if set to 1, EARL detected application loops will be reported to the database, each with its corresponding Signature. Set to 0 to disable this feature. Regardless of the setting, no error should occur.

If *Signatures* and/or *Periodic\_metrics* have additional columns but their respective settings are set to 0, a NULL will be set in those additional columns, which will make those rows smaller in size (but bigger than if the columns did not exist).



## Updating from previous versions

---

### From EAR 3.4 to 4.0

---

Several fields have to be added in this update. To do so, run the following commands to the database's CLI client:

```
ALTER TABLE Signatures ADD COLUMN avg_imc_f BIGINT unsigned AFTER avg_f;
ALTER TABLE Signatures ADD COLUMN perc_MPI DOUBLE AFTER time;
ALTER TABLE Signatures ADD COLUMN IO_MBS DOUBLE AFTER GBS;

ALTER TABLE Learning_signatures ADD COLUMN avg_imc_f BIGINT unsigned AFTER avg_f;
ALTER TABLE Learning_signatures ADD COLUMN perc_MPI DOUBLE AFTER time;
ALTER TABLE Learning_signatures ADD COLUMN IO_MBS DOUBLE AFTER GBS;
```

### From EAR 3.3 to 3.4

---

If no GPUs were used and they will not be used there are no changes necessary.

If GPUs were being used, type the following commands to the database's CLI client:

```
ALTER TABLE Signatures ADD COLUMN min_GPU_sig_id INT unsigned, ADD COLUMN max_GPU_sig_id INT un
ALTER TABLE Learning_signatures ADD COLUMN min_GPU_sig_id INT unsigned, ADD COLUMN max_GPU_sig_
CREATE TABLE IF NOT EXISTS GPU_signatures (id INT unsigned NOT NULL AUTO_INCREMENT, GPU_power
```

If no GPUs were being used but now are present, use the previous query plus the following one:

```
ALTER TABLE Periodic_metrics ADD COLUMN GPU_energy INT;
```

## CPU Models supported

---

- Intel Haswell/Skylake monitoring and optimization.
- AMD EPYC Rome monitoring.

## GPU models supported

---

- NVIDIA: Node and application monitoring.

## Schedulers supported

---

- EAR offers a SLURM SPANK plugin to be transparently used when using SLURM workload manager. This plug-in allows to be integrated as part of the SLURM submission options. See the user guide.
- Using the EARD api **new\_job/end\_job** functions EAR can be also be transparently used with other schedulers such as LSF or PBS through the prolog/epilog mechanism.

## CHANGELOG

---

### EAR 4.0

---

- AMD virtual p-states support and DF frequency management included
- AMD optimization based on min\_energy and min\_time
- GPU optimization in low GPU utilization phases
- Application phases IO/MPI/Computation detection included
- Node powercap and cluster powercap implemented: Intel CPU and NVIDIA GPUS tested. Meta EAR-GM not released
- IO, Percentage of MPI and Uncore frequency reported to DB and included in eacct
- econtrol extensions for EAR health-check

### EAR 3.4

---

- Automatic loading of EAR library for MPI applications (already in 3.3), OpenMP, MKL and CUDA applications. Programming model detection is based on dynamic symbols so it could not work if symbols are statically included.
- AMD monitoring support.
- TAGS support included in policies.
- Request dynamic in eard\_rapi.
- GPU monitoring support in EAR library for NVIDIA devices.
- Node powercap and cluster power cap under development.
- papi dependency removed.

### EAR 3.3

---

- eacct loop signature reported.
- EAR loader included.
- GPU support migrated to nvml API.
- GPU support in configure.
- TAGS supported in ear.conf.
- Heterogeneous clusters specification supported.
- EARGM energy capping management improved.
- Internal messaging protocol improved.
- Average CPU frequency and Average IMC frequency computation improved.

### EAR 3.2

---



- GPU monitoring based on nvidia-smi command.
- GPU power reported to the DB using NVIDIA commands.
- Postgresql support.
- freeipmi dependence removed.

## FAQS when using EAR flags with SLURM plugin

---

1) How to see EAR configuration and metrics at runtime? use `--ear-verbose=1` .

2) User authorized “issues”. The following list of ear flags are only allowed to Authorized users ( `ear.conf` ):  
`ear-cpufreq`, `ear-tag`, `ear-learning`, `ear-policy-th` .

**Action:** Check ear option and user authorization (`ear.conf`).

```
AuthorizedUsers=user1,user2
AuthorizedAccounts=acc1,acc2,acc3
AuthorizedGroups=xx,yy
```

If user is not authorized it means it is the expected result.

3) Why is a different energy policy other than the selected one being applied (validated with `--ear-verbose=1` )? The selected policy may not be enabled for all users. Energy policies can be configured to be enabled to all users or not.

**Action:** Check policy configuration (`ear.conf`) and user authorization (`ear.conf`).

```
#Enabled to all users
Policy=monitoring Settings=0 DefaultFreq=2.4 Privileged=0
#Enabled to authorized users
Policy=monitoring Settings=0 DefaultFreq=2.4 Privileged=1
```

If not enabled or not authorized it is the expected result.

4) How to disable EAR library explicitly: use `-ear=off` .

5) How to apply EAR settings to all **srun/mpirun** calls inside a job? Set options in `#SBATCH` headers.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -ear-policy=min_time
#application 1 and 2 will run with min_time
srun application1
srun application2
```

6) How to apply different EAR settings to different **srun/mpirun** calls inside a job? Set options for each step id.

```
srun -ear-policy=min_time application
srun -ear-policy=min_energy application
```

7) How to see which energy policies are installed? `srun --help`

**Comment:** Installed policies, it is possible a user is not allowed to run it.

8) How to set EAR flags with **mpirun (intel)**? Depending on the intel mpi version. Before version 2019, mpirun had 2 parameters to specify slurm options.

```
mpirun -bootstrap=slurm -bootstrap-exec-args="--ear-verbose=1"
```

Since version 2019, SLURM options must be specified using environment variables:

```
export I_MPI_HYDRA_BOOTSTRAP=slurm
export I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS --ear-verbose=1"
```

9) How to set EAR flags with **mpirun (openmpi)**? OpenMPI needs an extra support when srun is not used. EAR's **erun** command must be used.

```
mpirun erun -ear-policy=min_energy --program=application
```

10) An application is using OpenMPI and it blocks when running with EARL and **mpirun**: Use **erun**.

11) An application works without EAR (`--ear=off`) and fails with EARL reporting errors related with dynamic libraries:

**Action:** Check if the application is using right EAR mpi version. If environment variable is set in mpi modules, it must be automatic. Otherwise, validate whether `--ear-mpi-dist` is present when needed.

12) How to collect more detailed metrics than available in the DB. Use `--ear-user-db` flag to generate csv files with all EARL collected metrics.

13) How to collect paraver traces? Use the environment variables to enable the trace collection and to specify the path.

```
SLURM_EAR_TRACE_PLUGIN$EAR_INSTALL_PATH/lib/plugins/tracer/tracer_paraver.so
SLURM_EAR_TRACE_PATH=TRACES_PARAVER/
```

14) User asks for application metrics with **eacct** and NO-EARL appears in some of the columns in the output: This means EARL was not loaded with the application or the application fails before `MPI_Finalize`, nor reporting application data

**Action:** Check if application was executed with EARL and it didn't fail.

15) After some time, user asks for an application metrics with **eacct** and application is not reported.

**Action:** Try again after some minutes (applications are not reported immediately).