TABLE OF CONTENTS

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

1) A **easy-to-use and lightweight optimizarion 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 SLURM plugin (**EARPLUG, earplug.so**).

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.

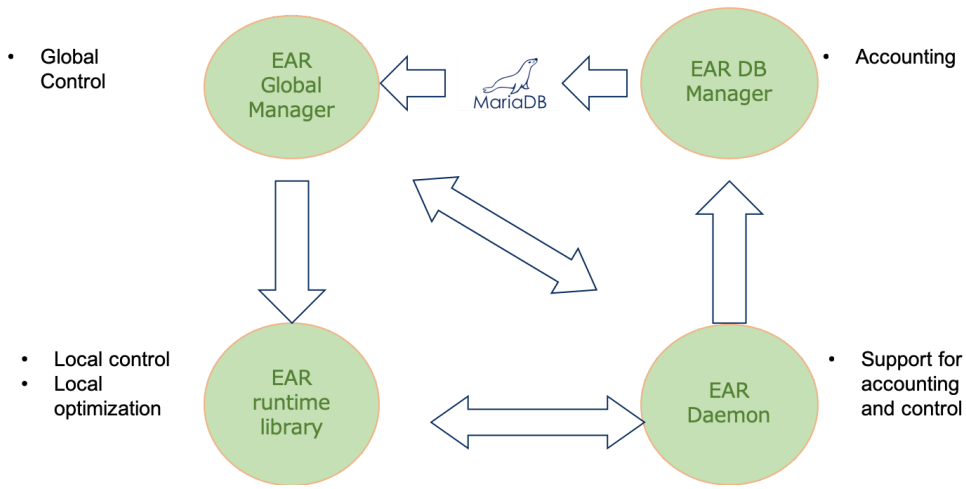
Contact: ear-support@bsc.es (mailto:ear-support@bsc.es)

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,papi, gsl, MPI, mysqlclient for mariaDB.
- To install EAR from rpm (only binaries) all these dependencies have been removed except mysqlclient. However, they are neeed 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.
- MySQL server must be up and running.

Installation and configuration

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 env vars 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 `$EAR_ETC/ear/ear.conf` and update with the desired configuration. Go to our ear.conf page to see how to do it.The ear.conf is used by all the services.
4. Load EAR module to enable commands. It can be found in `$EAR_ETC/module`. You can add ear module when it's not in standar 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 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 troubles, 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 EARDxa, and just for testings, it is needed to create a dummy coefficient file and copy in the coefficients path (by default placed at $EAR_ETC/coeffs). Visit the tools section, coeffs_null application.
7. Copy EAR service files to start/stop services using system commands such as systemctl. EAR service files are generated at `$EAR_ETC/systemd` and they can usually be placed in `$(ETC)/systemd`.

Execution and checks

1. 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´ respectivelly 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.
2. Check that the EARDs are up and running correctly with `econtrol --status` (note that the daemons will take around a minute to correctly report energy and not show up as an error in `econtrol`). EARDs creates a per-node text file with values reported to the EARDBD. In case there is problems when running econtrol, you can also find this file at `$EAR_TMP/nodename.pm_periodic_data.txt`.
3. Check that the EARDs are reporting metrics to database with `ereport` (`ereport -n all` should report the total energy send by each daemon since the setup).
4. Start EARGM via services.
5. 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`, option GlobalManagerPeriodT1, to report for the first time.).
6. Set up EAR's SLURM plugin (see our Configuration page for more information).
7. 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).

8. 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 it is specified in MPI_VERSION during the configure. When installed from rpm, look at ´$EAR_INSTALL_PATH/lib´ to see the available versions.
9. Set `default=on` to specify the EAR library will be loaded with all the applicatins 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.
10. 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.
11. 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
PAPI	5.4.0	Website (http://icl.utk.edu/papi/)
GSL	1.4	Website (https://www.gnu.org/software/gsl/)
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)

- Just one of them required.

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 compiler and Fortran compiler. 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

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 `$(EAR_TMP)` in this guide for simplicity).
2. Generate Autoconf's `configure` program by typing `autoreconf -i`.
3. Compile EAR components by typing `./configure`, `make` and `make install` in the root directory. Consider the option of `./configure --PREFIX=<path>` if you want to specify the installation path. It could be useful to run './configure --help' for listing the options details. You can install individual components by doing: `make eard.install` (install eard), `make earl.install` (ear library), `make eardbd.install` (eardbd), `make eargmd.install` (eargm) and `make commands.install` (ear commands)
4. Type `make etc.install` to install the content of `$EAR_ETC`. It is a configuration content, but that configuration will be expanded in the next section.

You can customize the installation by reading the below sections.

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 alo use --localstatedir=DIR).
EAR_ETC	Defines the read-only single-machine data as 'etc' (default: EPREFIX/etc) (you can also use --sharedstatedir=DIR).
MAN	Defines the manual directory (default: PREFIX/man) (you can use also --mandir=DIR).
DOC	Defines the documentation directory (default: PREFIX/doc) (you can use also --docdir=DIR).
MPI_VERSION	Adds a suffix to the compiled EAR library name. Read on 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 PAPI, 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
--with-papi=\<path>	Specifies the path to PAPI installation.
--with-gsl=\<path>	Specifies the path to GSL installation.
--with-slurm=\<path>	Specifies the path to SLURM installation.
--with-mysql=\<path>	Specify path to MySQL installation.
--with-pgsql=\<path>	Specify path to PostgreSQL installation.
--with-fortran	Adds Fortran symbols to the binaries. Required for some MPI distributions.

- This is an example of 'CC' overwriting and PAPI path specification:
  `./configure --with-papi=/path/to/PAPI`

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 the nodes.

Argument	Description
--disable-rpath	Disables the RPATH included in binaries to specify some dependencies location.
--disable-avx512	Replaces the AVX-512 function calls by AVX-2.

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-procesor 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 the root Makefile.

Adding multiple library distributions/versions

As commented in the overview, the EAR library is loaded next to the user MPI application. If the cluster provides different MPI distributions or versions, different library versions have to be compiled to match with that MPI symbols.

To do this easily, you can run the `./configure` again, or just open the root `Makefile` and edit the `CC` and `MPICC` compilers. Also, the `MPI_VERSION` variable has to be named to add a suffix, for example `MPI_VERSION = intel2019` to generate the final binary `libear.intel2019.so` , avoiding a file replacement when installing again.

If you compiled previously, type `make full` to clean and compile all files again. Take into account that when cleaning, the previous compilation binaries are removed, so type `make install` after that each time you compile.

If you added a suffix to all the compiled libraries, a softlink named `libear.so` is required to define the default library. If a user do not select any specific library, the default one will be loaded. In the user guide, it is commented how to select the version before running a user application.

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.plugstack.conf.
\<*EAR_ETC*>	/systemd	EAR service files.

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. Depending on the RPM, different version must be required of this libraries:

Library	Required / comment	References	Dependency
PAPI	Yes	Website (http://icl.utk.edu/papi/)	EAR library, when running
GSL	Yes	Website (https://www.gnu.org/software/gsl/)	coeffs_compute
MPI	Yes	-	EAR library, when running
MySQL	Yes	-	RPM installation

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

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)

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.plugstack.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.plugstack.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. `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. 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: EAR saves data in a MySQL DB server. EAR DB can be created using `edb_create` command provided (MySQL 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.

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

MariaDB configuration

```
# The IP of the node where the MariaDB (MySQL) or Postgress server process is running.
MariaDBIp=172.30.2.101
# Port in which the server accepts the connections.
MariaDBPort=3306
# MariaDB user that the services will use. Needs INSERT/SELECT privileges. Used by EARD

MariaDBUser=eardbd_user
# Password for the previous user. If left blank or commented it will assume the user ha
MariaDBPassw=eardbd_pass
# MariaDB user that the commands (eacct, ereport) will use. Only uses SELECT privileges
MariaDBCommandsUser=ear_commands
# Password for the previous user. If left blank or commented it will assume the user ha
MariaDBCommandsPassw=commandspass
# Name of EAR's database in the server.
MariaDBDatabase=EAR
# Maximum number of connections of the commands user to prevent server saturation/malic
MaxConnections=20
# The following specify the granularity of data reported to database.
# Extended node information reported to database (added: temperature and avg_freq in pc
ReportNodeDetail=1
# Extended signature hardware counters reported to database.
ReportSigDetail=1
# Set to 1 if you want Loop signatures to be reported to database.
ReportLoops=0
```

EARD configuration. EARD are executed in compute nodes

```
# The port where the EARD will be listening.
NodeDaemonPort=5000
# 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
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
NodeUseLog=1
# Minimum time between two energy readings for performance accuracy
MinTimePerformanceAccuracy=10000000
```

EARDBD configuration

```
# Port where the EARDBD server is listening
DBDaemonPortTCP=4711
# Port where the EARDBD mirror is listening
DBDaemonPortSecTCP=4712
# Port is used to synchronize the server and mirror
DBDaemonSyncPort=4713
# 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
DBDaemonMemorySize=120
# The percentage of the memory buffer used by the previous field, by each type. These t
DBDaemonMemorySizePerType=40,20,5,24,5,1,5
# 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=4
# 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
DynaisTimeout=15
# Time in seconds to compute every application signature when the EAR goes to periodic
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 demon is running.
GlobalManagerHost=hostname
# Port where EARGMD will be listening.
GlobalManagerPort=50000
# Use '1' or not '0' aggregated metrics to compute total energy.
GlobalManagerUseAggregated=1
# Period T1 and period T2 are specified in seconds. T1 must be less than T2. Global mar
GlobalManagerPeriodT1=90
GlobalManagerPeriodT2=259200
# Units field, Can be '-' (Joules), 'K' KiloJoules or 'M' MegaJoules
GlobalManagerUnits=K
# This limit means the maximum energy allowed in 259200 seconds in 550000 KJoules
GlobalManagerEnergyLimit=550000
#
GlobalManagerPolicy=MaxEnergy
# Global manager modes. Two modes are supported '0' (manual) or '1' (automatic). Manual
GlobalManagerMode=0
# A mail can be sent reporting the warning level (and the action taken in automatic moc
GlobalManagerMail=nomail
# Percentage of accumulated energy to start the warning DEFCON level L4, L3 and L2.
GlobalManagerWarningsPerc=85,90,95
# Number of "grace" T1 periods before doing a new re-evaluation. After a warning, EARGM
GlobalManagerGracePeriods=6
# Verbose level
GlobalManagerVerbose=1
# When set to 1, the output is saved in '$EAR_TMP'/eargmd.log (common configuration) as
GlobalManagerUseLog=1
```

Common configuration

```
# Network extension (using another network instead of the local one). If compute nodes
# NetworkExtension=netext
# Default verbose level
Verbose=0
# Path used for communication files, shared memory, etc. It must be PRIVATE per compute
TmpDir=/tmp/ear
# Path where coefficients and configuration are stored. It must be readable in all comp
EtcDir=/path/to/etc
InstDir=/path/to/inst
# Path where metrics are generated in text files when no database is installed. A suffi
DataBasePathName=/etc/ear/dbs/dbs.
```

Power policies plugins

```
# Policy names must be exactly file names for policies installed in the system at /path
DefaultPowerPolicy=min_time
# Example of the definition of 3 policies with different configurations: It must be inc
Policy=monitoring Settings=0 DefaultFreq=2.3 Privileged=0
Policy=min_time Settings=0.7 DefaultFreq=2.0 Privileged=0
Policy=min_energy Settings=0.1 DefaultFreq=2.3 Privileged=0
```

Other plugins

```
# Energy reading plugin (without the extension). Allows to use different system compone
# look at /path/to/inst/lib/plugins/energy folder to see the list of installed energy p
PluginEnergy=energy_nm
# Power model plugin (without the extension). The power model plugin is used to predict
PluginPowerModel=default
```

Security

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 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=acc
```

Special nodes

Describes nodes with some special characteristic such as default coefficients file

```
NodeName=nodename_list CPUs=24 DefCoefficientsFile=filenam
```

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 line describes an island, and every node must be in an island.

Remember that 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 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.

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 a island to the server and the mirror 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.

The `min_power`, `max_power` and `max_temp` 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.

```
Island=0 Nodes=nodename_list DBIP=EARDB_server_hostname DBSECIP=EARDB_mirror_hostname m
```

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`

Please visit the islands example for more information and examples of a cluster configuration in form of islands.

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_pa
```

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, and `eargmd_host` and `eargmd_port`, if you plan to connect with the EARGMD component (you can leave this empty).

MySQL

WARNING: If any EAR component is running in the same machine as the MySQL server some connection problems might occur. 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 MySQL 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/MariaDB'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 run.

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.

Components execution

The best way to execute all EAR daemon components (EARD, EARDBD, EARGM) is by the unit services method.

NOTE: EAR uses a MariaDB/MySQL server. The server must be started before EAR services are executed.

Unit services

The generated unit services for the EAR Daemon, EAR Global Manager Daemon and EAR Database Daemon are generated and installed in `$(EAR_ETC)/systemd`. You have to copy those unit service files to your `systemd` operating system folder ant then use the `systemctl` command to run the daemons.

Launching the components through unit services

The way to launch the EAR daemons is by the unit services method. The generated unit services for the EAR Daemon, EAR Global Manager Daemon and EAR Database Daemon are generated and installed in `$(EAR_ETC)/systemd`. You have to copy those unit service files to your `systemd` operating system folder and then use the `systemctl` command to run the daemons.

Check the EARD, EARDBD, EARGMD pages to find the precise execution commands.

Finally, when using `systemctl` commands, you can check messages reported to `stderr` using `journalctl`. For instance: `journalctl -u eard -f`. Note that if `NodeUseLog` is set to 1 in `ear.conf`, the messages will not be printed to `stderr` but to `$EAR_TMP/eard.log` instead. DBDaemonUseLog and GlobalmanagerUseLog options in ´ear.conf´ specifies the output for EARDBD and EARGM respectivelly.

Additionally, services can be started, stopped or reloaded on parallel using parallel commands such as pdsh. As an example: `sudo pdsh -w nodelist systemctl start eard`

Tests

The EAR package includes two type of tests. The **check tests**, prepared to be executed after the `make` by typing `make check`, with and without privileges, so probably you will have to check it with `sudo`.

When running a **check test**, 3 types of message are written in the output. 1) **Error**: the hardware, software or libraries are incompatible with the library. 2) **Warning**: it's possible that some componentes have to be loaded prior the execution (the library would try to do it). 3) **Ok**: your system is full compatible with the library.

Make check tests list

Name	Checking
cpu_examinable	If the CPU examinable by the library.
cpu_aperf	If the CPU APERF counter is available.
cpu_uncores	If there are CPU uncore counters available.
cpu_uncores_all	If there are all CPU uncore counters available.
papi_version	If the PAPI version is greater or equal than the reference.
papi_init	If PAPI initializes correctly.
papi_comp_available	If PAPI perf counters events are available.
papi_comp_enabled	If PAPI perf counters events are enabled.
papi_comp_available	If PAPI perf uncore counters events are available.
papi_comp_enabled	If PAPI perf uncore counters events are enabled.
papi_comp_available	If PAPI libmsr events are available.
papi_comp_enabled	If PAPI libmsr events are enabled.
papi_comp_available	If PAPI rapl events are available.
papi_comp_enabled	If PAPI rapl events are enabled.
gsl_version	If the GSL version is greater or equal than the reference.
slurm_version	If the SLURM version is greater or equal than the reference.
module_ipmi_devintf	If the `ipmi_devintf` (IPMI) driver is running.
module_acpi_cpufreq	If the `acpi-cpufreq` (CPUFreq) driver is running.

Tools list

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

- 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

```
/compute_coeffs /etc/coeffs 1900000 node1001
```

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 configurarion 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 disables 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    verbose mode for debugging purposes
                -u    specifies the user whose applications will be retrieved. Only ava
                -j    specifies the job id and step id to retrieve with the format [job
                        A user can only retrieve its own jobs unless said user is privi
                -c    specifies the file where the output will be stored in CSV format.
                -t    specifies the energy_tag of the jobs that will be retrieved. [def
                -l    shows the information for each node for each job instead of the g
                -x    shows the last EAR events. Nodes, job ids, and step ids can be sp
                -n    specifies the number of jobs to be shown, starting from the most
                -f    specifies the file where the user-database can be found. If this
```

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
JOB-STEP  USER APPLICATION POLICY NODES#  FREQ(GHz)  TIME(s)    POWER(Watts)   GBS
31191-5      user bqcd_cpu    ME     49      2.24       404.57     217.25         4.19
31191-4   user bqcd_cpu    ME     50      2.27       398.38     229.09         4.26
31191-3   user bqcd_cpu    ME     50      2.28       394.89     230.84         4.30
```

Columns shown are: job id.stepid, username, application name, policy (NP means EAR Library was not loaded)), number of nodes, average frequency, execution time, average power, GBs, Cycles per instruction (CP),energy, GFlops/Watt and Maximum Power.

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 cons
    -e end_time            indicates the end of the period from which the energy consum
    -n node_name |all      indicates from which node the energy will be computed. Defau
                   'all' option shows all users individually, not aggregated.
    -u user_name |all      requests the energy consumed by a user in the selected perio
                   'all' option shows all users individually, not aggregated.
    -t energy_tag|all      requests the energy consumed by energy tag in the selected p
                   'all' option shows all tags individually, not aggregated.
    -i eardbd_name|all     indicates from which eardbd (island) the energy will be comp
                   'all' option shows all eardbds individually, not aggregated.
    -g                     shows the contents of EAR's database Global_energy table. Th
                  This option can only be modified with -s, not -e
    -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 point 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 custer'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    ENE
111.486          100  2019-05-22 10:31:34            0          100          893      1
111.492          100  2019-05-22 10:21:34            0          100          859      1
111.501          100  2019-05-22 10:11:34            0          100          862      1
111.514          100  2019-05-22 10:01:34            0          100          842      1
111.532          100  2019-05-22 09:51:34            0          100          828      1
111.554            0  2019-05-22 09:41:34            0            0          837      1
```

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]
    --set-freq     newfreq           ->sets the frequency of all nodes to the requeste
    --set-def-freq     newfreq    policy_name    ->sets the default frequency for the s
    --set-max-freq     newfreq           ->sets the maximum frequency
    --inc-th     new_th    policy_name    ->increases the threshold for all nodes
    --set-th     new_th    policy_name    ->sets the threshold for all nodes
    --red-def-freq     n_pstates        ->reduces the default and max frequency by n ps
    --restore-conf                ->restores the configuration to all node
    --status                 ->requests the current status for all nodes. The ones resp
                          power, IP address and policy configuration. A list with the
                          responding is provided with their hostnames and IP address.
                          --status=node_name retrieves the status of that node indivi
    --ping                  ->pings all nodes to check wether the nodes are up or not
                          --ping=node_name pings that node individually.
    --help                   ->displays this message.
```

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

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://github.com/BarcelonaSupercomputingCenter/ear_learning/wiki) to read the manual and the repository (https://github.com/BarcelonaSupercomputingCenter/ear_learning) to get the scripts and the kernels.

EAR plugins

Some of the core EAR functionality can be dynamically loaded through a plugin 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 plugins are: the node energy reading library, the power policy, the power model, and the tracing.

Plugin	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 betwen 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

- Plugin paths is set by default to $EAR_INSTALL_PATH/lib/plugin.
- Default power model library is specified in ´ear.conf´(PluginPowerModel option). by default EAR includes a ´basic_model.so´ library.
- The node energy readings library is specified in ´ear.conf´ in the PluginEnergy 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: min_energy.so, min_time_no_models.so monitoring.so min_energy_no_models.so min_time_avx512.so min_time.so. The list of policies installed is automatically detected by the EAR plugin. However, only policies included in ear.conf can be used.
- The tracing is an optional functionality. It is included to provided additional information or to generate runtime information
- Note: SLURM Plugin does no fit in this philosophy, it is a core component of EAR and can not replaced by third party development.

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 so please, check the configuration page for more information.

Plugin `min_energy`

The goal of this policy is to minimize the energy consumed with a limit to the performance degradation. This limit is is set in the SLURM option or the configuration file. The `min_energy` policy will select the optimal frequency that minimizes energy enforcing (performance degradation <= parameter). When executing with this policy, applications starts at nominal frequency.

```
PerfDegr = (CurrTime - PrevTime) / (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 said frequency increment. The policy uses the parameter option 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% (PerfGain >= (FreqGain * threshold).

```
PerfGain=(PrevTime-CurrTime)/PrevTime
FreqGain=(CurFreq-PrevFreq)/PrevFreq
```

When executed with `min_time` policy, applications starts at a default predefined frequency lower than nominal (defined at `ear.conf` , check the configuration page for more information.

Example: given a system with a nominal frequency of 2.3GHz and default frequency set to 3, an application executed with `min_time` 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 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 threashold, the policy will select the last frequency where the performance gain was enough, preventing the waste of energy.

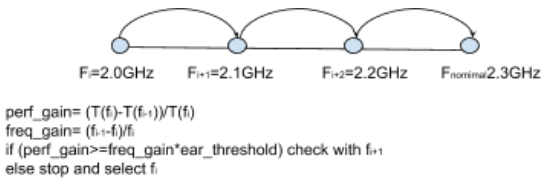


Figure 1: `min_time` uses threashold as the minimum value for the performance gain between between `F[i]` and `F[i+1]`.

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.

Job submission with EAR

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

Options	Description
--ear=on/off(**)	Enables/disables EAR library.
--ear-policy=policy	Selects an energy policy for EAR. See the Policies page for more info
--ear-cpufreq=frequency(*)	Specifies the starting frequency to be used by the chosen EAR policy (in KHz).
--ear-policy-th=value(*)	Specifies the ear_threshold to be used by the chosen EAR policy { `value=[0...1]` }.
--ear-user-db=file	Specifies the files where the user applications' metrics summary will be stored {'file.nodename.csv'}. If not defined, these files will not be created.
--ear-mpi-dist=dist(***)	Selects the library distribution/version for compatibility of the application {dist=intel,intel2019,openmpi,...}.
--ear-verbose=value	Specifies the level of verbosity {value=[0...2]}; the 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]` }.

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. (***) libear.so is used by default. SLURM_EAR_MPI_VERSION env var can be also used to specify the version (for instance, to be defined in mpi modules)

Examples

`srun` examples

EAR plugin reads `srun` options and contacts with EARD. Invalid options are filtered to default values, so behaviour 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 authorised, job will be executed at 2.0GHz as default frequency and with power policy set to min_time. Otherwise, default values will be applied:

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

- Executes application with EAR. If users is authorised 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=
```

`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
#SBTACH --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)

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 following example will run application with min_time_to_solution policy:

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

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

User commands

The only command available to users is `eacct`. With `eacct` 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 a number of 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 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)`
- `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),
             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();
}
```

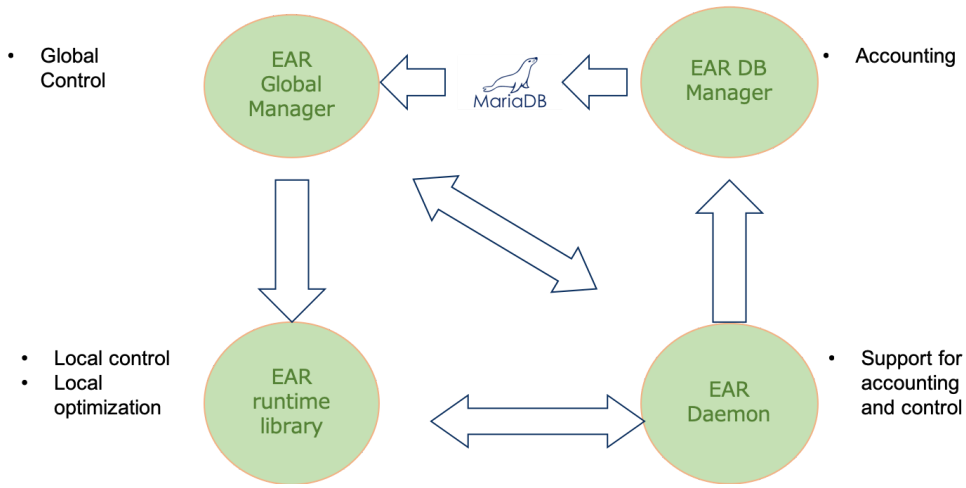
License

All the files in the EAR framework are under the LGPLv2.1 license. See the COPYING file in the EAR root directory.

EAR is composed of five main components:

- Node Manager (EARD)
- Database Manager (EARDBD)
- Global Manager (EARGM)
- Library (EARL)
- 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 3.1 - EARD.

Database Manager

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

For more information, see 3.2 - 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 3.3 - 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 3.4 - EARL.

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 3.5 - 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 two 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, 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.

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, 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`

Reconfiguration

After executing a "systemctl reload eard" command, not all the EARD options are 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.

API

The (node) Daemon offers a simple API to request changes on the frequency, modify the current node settings, and reload the system configuration by reading `$(EAR_ETC)/ear/ear.conf`

Three APIs are provided:

- Local API, to be used by EARL (or any other runtime). It can be found in eard_api.h. This API involves complex data types and is not public.
- Local API, to be used by applications. It is a subset of the EARD api and designed to be used by any applications to contact the privileged metric service offered by EARD. This API is public and can be used without restrictions, therefore it does not include functions to change the frequency. It can be found at TBD.
- Remote API, to be used by the EARGMD or system commands and tools such as the `econtrol`. Can be found at eard_rapi.h and is not public.

EARDBD: Database Manager

EARDBD caches the 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, 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, 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, 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.

EARGM: Global Manager

EARGM is a cluster wide component that controls the percentage of the maximum energy consumed. It can be configured to take actions automatically like warning sysadmins to take actions or limiting the nodes policy.

EARGM uses periodic power metrics reported by EARD, the per-node daemon, including job identification details (job id and step id if you are using the SLURM plugin). These metrics are stored and aggregated in a MariaDB (MySQL) database through the EARDBD.

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, this `systemctl` command examples are provided:

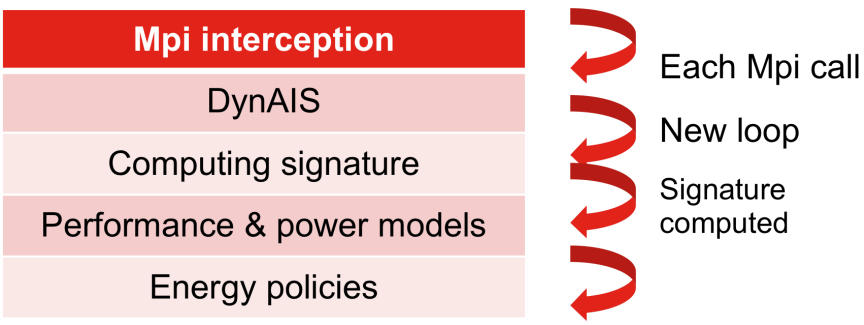
- `sudo systemctl start eargmd` to start the EARGM service.
- `sudo systemctl stop eargmd` to stop the EARGM service.
- `sudo systemctl reload eargmd` to force to reload the configuration of the EARGM service.

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

At runtime, EARL goes through the following phase:



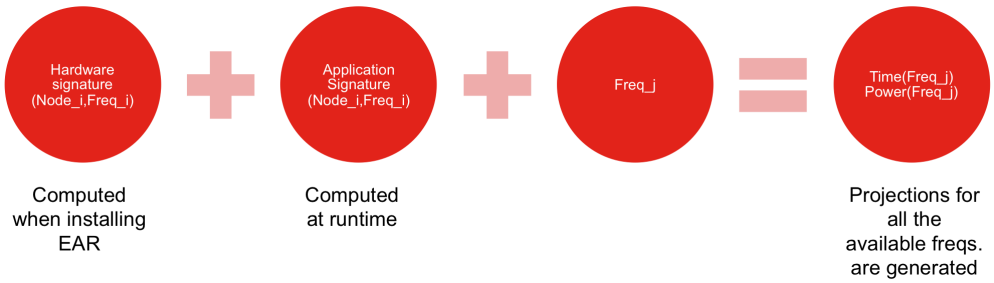
1. Automatic detection of application outer loops. This is done by dynamically intercepting MPI calls (using `LD_PRELOAD`) and invoking the Dynamic Application Iterative Structure detector algorithm. DynAIS is highly optimized for new Intel architectures, reporting low overhead.
2. Computation of the 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.

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

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

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

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 at the learning phase at the EAR configuration time. 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 this application and this particular run. An additional policy, 'monitoring only' can also be used, but in this case no changes to the running frequency will be made and only the computation of the application signature and metrics storing 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 the its specific settings through a shared memory regiones initialized by EARD.

How to run MPI applications with EARL

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

SLURM plugin configuration guide

EAR SLURM plugin allows to dynamically load the EAR library for the SLURM jobs, if the enabling argument is set or is enabled by default. The library will be loaded in each job step, intercepting all MPI calls.

Configuration

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

ERUN

ERUN is a program that simulates all the SLURM and EAR SLURM Plugin pipeline. If a set of nodes does not have SLURM installed, but they have MPI or other job manager, you can launch ERUN instead your application directly. In example:

```
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.
    --plugstack [ARGS]    Set the SLURM's plugstack arguments. I.e:
        --plugstack prefix=/hpc/opt/ear default=on...
    --clean        Removes the internal files.

SLURM options:
...
```

The `--job-id` and `--nodes` parameters, creates 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.

Finally, the `--plugstack` options sets the list of `plugstack.conf` parameters by the ERUN input, because that configuration file is not read by SLURM (because currently there is no SLURM). In example:

```
mpirun -n 4 /path/to/erun --program="hostname --alias" --plugstack prefix=/hpc/opt/ear
```

You can see a complete list of `plugstack.conf` parameters in the configuration page.

Instead passing as an input the plugstack parameters, you can define some environment variables with the same functionality:

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>

This allows you to write it just one time in scripts, in user environment or in a module.

Lastly, the tipical 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
```

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 every N seconds (N 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 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**: 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 the 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

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:

- `ReportNodeDetail` : if set to 1, `edb_create` will create to additional columns in the Periodic_metrics table for Temperature (in Celsius) and Frequency (in Hz) accounting.
- `ReportSigDetail` : if set to 1, Signatures will have additional fields for cycles, instructions, and FLOPS1-8 counters (number of instruction by type).
- `MaxConnections` : 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.

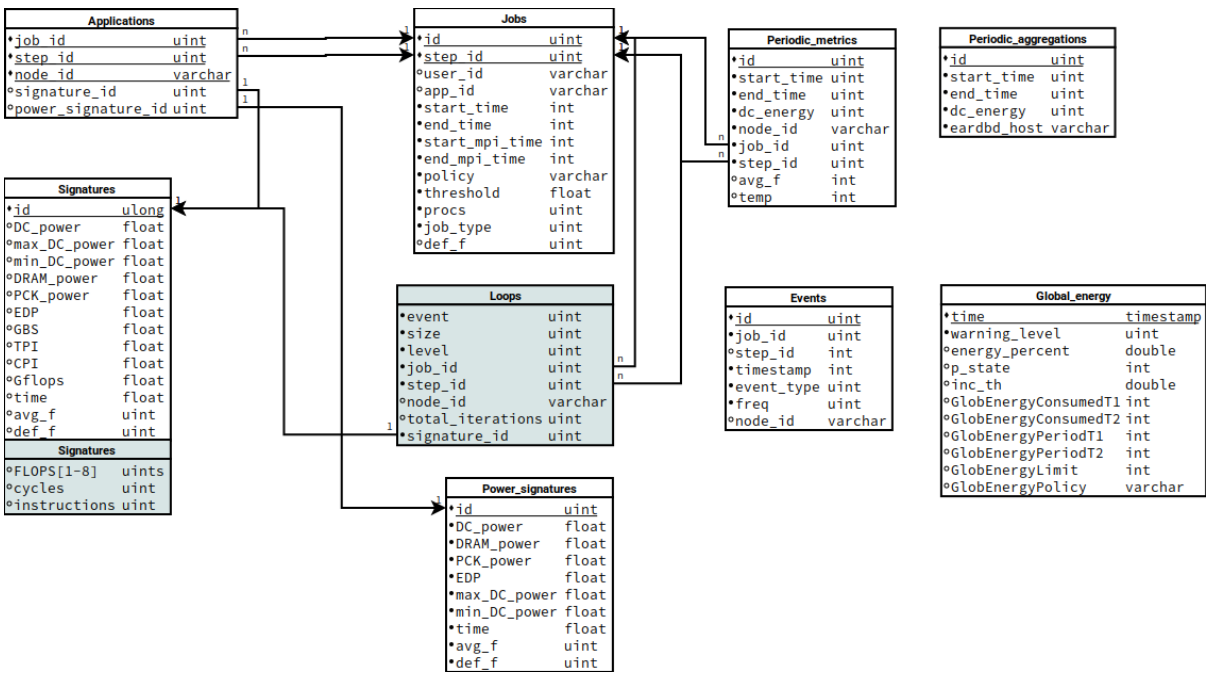
Any table with missing columns can be later altered by the admin to include said columns. For a full detail of each table's columns, run `edb_create -o` with the desired `ear.conf` settings.

Information reported and `ear.conf`

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

- `ReportNodeDetail` : if set to 1, the node managers will report temperature and average frequency 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 to have the necessary columns.
- `ReportSigDetail` : similarly to `ReporNodeDetail` , an error will occur if the configuration differs from the one used when creating the database.
- `ReportLoops` : if set to 1, EARL detected application loops will be reported to 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 the additional columns but their respective settings are set to 0, a NULL will be set in said additional columns, which will make those rows smaller in size (but bigger than if the columns did not exist).



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) How to select a specific energy policy and a different one is applied (validated with ear-verbose=1). 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 enables 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 the srun/mpiruns inside a job: Set the 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 eat settings to different srun/mpirun inside a job: The options per stepid.

```
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 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. Erun command must be used.

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

10) Application is using OpenMPI and it blocks when running with EARL and mpirun: Use erun

11) Application works without EAR (--ear=off) and fails with EARL reporting errors related with dynamic libraries

Action: Check if application is using right EAR mpi version. If environment variable is set in mpi modules, it must be automatic, otherwise, validate the --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 the 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 , that 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 application metrics with eacct and application is not reported

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