Project Title	Hybrid Eco Responsible Optimized European Solution
Project Acronym	HEROES
Grant Agreement No.	956874
Start Date of Project	01.03.2021
Duration of Project	24 Months
Project Website	heroes-project.eu

D4.1 – Updated Energy Aware Runtime

Work Package	**WP 4, Optimisation: Energy management & Decision module**
Lead Author (Org)	**Julita Corbalan (BSC)**
Contributing Author(s)	
Reviewed by	**Benjamin Depardon (UCit), Anaëlle Dambreville (UCit), Elisabeth Ortega (HPCNow),**
Approved by	**Name (organization)**
Due Date	**01.09.2022**
Date	**13.10.2022**
Version	**V1.2**

Dissemination Level

X	PU: Public
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

Versioning and contribution history

Version	Date	Author	Notes
0.1	19.09.2022	Julita Corbalan (BSC)	TOC and V0.1
0.2	20.09.2022	Elisabeth Ortega (HPCNow!)	Review
0.3	22.09.2022	Anaelle Dambreville (UCit)	Review
0.4	23.09.2022	Benjamin Depardon (UCit)	Review
1.0	05.10.2022	Julita Corbalan (BSC)	Final review
1.1	06.10.2022	Corentin Lefevre (Neovia)	Edition
1.2	13.10.2022	Corentin Lefevre (Neovia)	Final version approved by the Management Board

Disclaimer

This document contains information which is proprietary to the HEROES Consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to a third party, in whole or parts, except with the prior consent of the HEROES Consortium.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

Table of Contents

Executive Summary 5

1 Introduction 6

2 EAR Overview: services and functionalities 6

3 EAR execution on HEROES scenarios 8

3.1 HEROES scenarios 8

4 EAR extensions for HEROES 9

4.1 Automatic deployment 9

4.2 Initialization 10

4.3 Support for execution in scenarios where EAR is not installed 12

4.4 HEROES report plugin 13

4.5 Singularity 16

4.6 Cloud 18

5 Summary 18

6 References 19

List of Figures

Figure 1 EAR : System software for energy management 6
Figure 2 EAR components 7
Figure 3 Default EAR architecture 8
Figure 4 EARD/EARL interaction in a full installation 10
Figure 5 EARL behaviour in the absence of EARD 11
Figure 6 FLOPS API example 13
Figure 7 Report plugin API 14

List of Tables

Table 1 Application header 15
Table 2 Exclusive headers for loops 16

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

TERMINOLOGY

Terminology/Acronym	Description
API	Application Programming Interface
AWS	Amazon Web Service
CPI	Cycles per Instruction
CPU	Central Processing Unit
DB	DataBase
EAR	Energy Aware Runtime
EARD	EAR Daemon
EARDBD	EAR Database Daemon
EARGMD	EAR Global Manager Daemon
EARL	EAR Library
GPU	Graphics Processing Unit
HPC	High-Performance Computing
MPI	Message Passing Interface
OS	Operating System

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

Executive Summary

This deliverable describes the main challenges on the Energy Aware Runtime (EAR) to support different HPC infrastructures and scenarios. EAR is a system software for energy management for HPC Data Centres. Before the HEROES project, all the EAR components were designed assuming their execution on a full EAR deployment and in classic bare-metal HPC infrastructures. However, given the goals of the HEROES project, this assumption was no longer valid.

This document includes, together with an EAR overview, the description of the EAR requirements to be executed on the HEROES scenarios and the design and implementation of the technical solutions to deal with the potential limitations depending on the infrastructure.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

1 Introduction

EAR is a system software for energy management [**3**] [**4**]. Its services and functionalities can be represented as shown in Figure 1. On the bottom there is a powerful infrastructure for energy and performance monitoring. This feature is divided in two modules. The first one, (1) in the figure, provides basic job accounting and system monitoring. Basic job accounting includes power/energy and execution time for all the jobs executed in the system. This service is provided in cooperation with the job scheduler. The system monitoring includes performance and power metrics from a hardware perspective. The second module, (2) in the figure, is an extended performance and power monitoring with application semantics. On top of this second module together with node monitoring, EAR builds the application and cluster energy optimization and power management. This third module, (3) in the figure, includes energy optimization policies applied at runtime to running jobs and cluster powercapping.

Figure 1 EAR : System software for energy management

These three modules are implemented using low level features such as hardware performance counters, OS metrics and job scheduler support. Given EAR software is supposed to be executed in clusters where EAR is deployed, if some of these features are not working in one particular compute node or for one application, it is disabled. This simplification was good enough for clusters where these issues are supposed to be the exception. Next section enters into more details about main EAR components.

2 EAR Overview: services and functionalities

The *Energy Aware Runtime* (EAR for short) is a package that provides an energy management and monitoring framework for High Performance Computing (HPC) centres. The EAR package consists of five main components (see Figure 2):

- The EAR Library (EARL for short) is a smart component loaded next to the application, which intercepts MPI calls and selects CPU frequencies on the fly depending on the behaviour of the application in accordance to the selected policy. The EARL

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

optimization policies as well as a basic EAR software overview can be found in [1] and [2].

- The EAR Daemon (EARD for short) is the only component that requires root privileges, and is in charge of measuring metrics and actually setting the CPU frequencies set by the EAR Library. There is one EAR Daemon loaded per compute node.
- The EAR Database Daemon (EARDBD for short) is the component in charge of collecting and reporting the metrics obtained during execution. It uses a plugin system to report metrics to a database, with MariaDB and PostgreSQL plugins currently available with the main distribution. There is an EARDBD per service node.
- The EAR Global Manager Daemon (EARGMD for short) is an optional component that overviews the execution throughout the whole system, and is able to dynamically adapt the policy settings of the different EARL applications running in the system to control the maximum power used at any certain time by the cluster.
- The EAR Plugin, which is a SLURM plugin meant to send the job information to the EARL on running a new job.

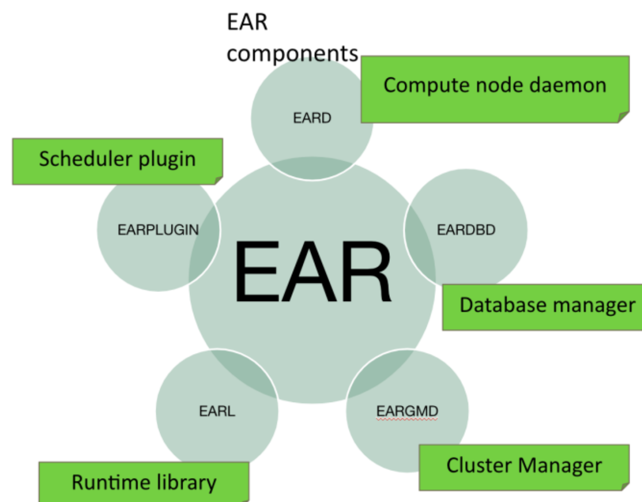

Figure 2 EAR components

EAR is supposed to be deployed as shown in Figure 3. One instance of the EARD is executed in each of the compute nodes. The EARD provides support for privileged metrics and capabilities that requires privilege access such as changing the CPU frequency. The scheduler plugin notifies the EARD about the execution of jobs, sending messages for each new job and termination of job. Using these messages, the EARD can provide job energy accounting for all the jobs executed in the system.
The loading of the EARL is automatically done with the job scheduler support. In particular, EARL includes a SPANK plugin for SLURM [5], making 100% transparent the job submission with EAR.
The EARL implements the energy optimization policies with the EARD support and reports to the DB all the metrics collected at runtime with the EARDBD support. These metrics are called application and runtime signature (or loop signature). The signature includes performance and power models used by the EARL to classify application activity and to compute the time

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

and power models used for the energy policies. Data is reported by default to a relational DB using the support of the EARD and EARDBD, but the EARL uses an extensible report API supporting multiple targets at the same time.
The EARDBD (or EARDBDs) gathers data to be reported to the DB and optimizes the number of connections and messages sent to the DB by implementing some buffering.

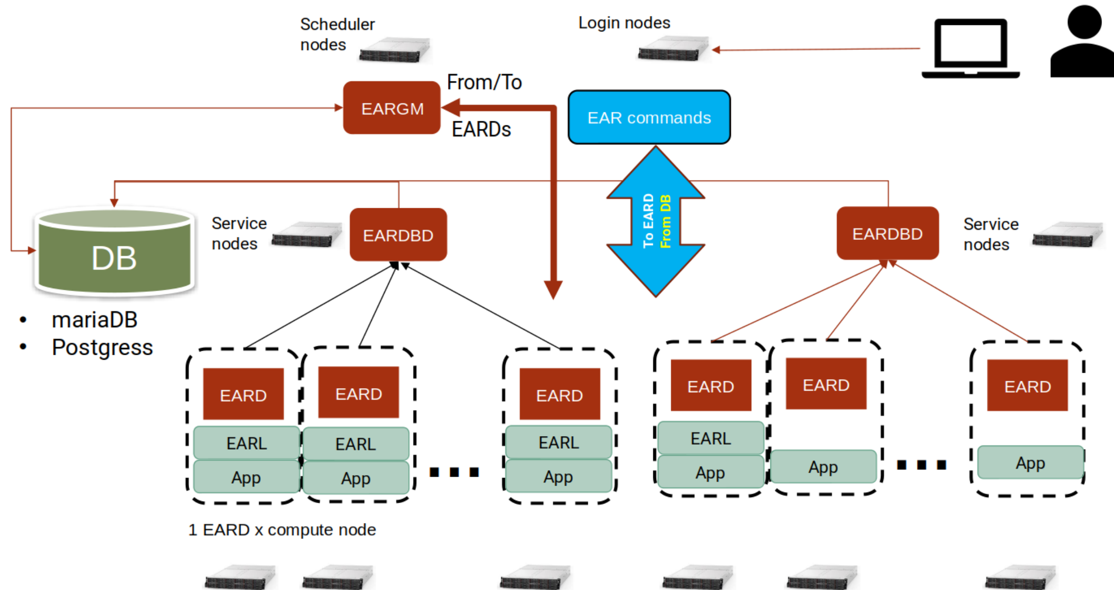

Figure 3 Default EAR architecture

Finally, the EARGM, not involved in this project, is the component implementing cluster-wide services such as cluster energy optimization and cluster powercap.

3 EAR execution on HEROES scenarios

The deployment and execution of EAR in HEROES scenarios presents two main challenges. Concerning EAR deployment, the requirement is to be 100% automatic and concerning EAR execution, it is to be 100% usable independently of the underlying hardware. Of course, we are not targeting to be able to provide the same services and capabilities than in a fully deployed system, but to be able to provide as much services as possible.

We will refer to the execution of the EARL in these limited scenarios as EAR Lightweight (or EAR Light), since a partial support of EAR features will be provided.

3.1 HEROES scenarios

The following are the main scenarios contemplated when integrating EAR in the HEROES project:

- HPC centres where EAR is in production (fully deployed). In this case, we would like to support, at least, these two types of jobs:
    - Traditional HPC jobs: Normal state, where users launch a job through the job scheduler and EAR is fully operational.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

  - Jobs executed with Singularity: Similar to the previous case, but the entire environment is virtualized through Singularity.
- HPC centres where EAR is not installed:
  - Traditional HPC jobs: Users still launch a job through the job scheduler, but EAR is missing some key components (EARD) and it needs to operate with reduced functionality.
  - Jobs executed with singularity containers [**6**] : Similar to the previous case, but the entire environment is virtualized through Singularity.
- Cloud: Data centres or HPC centres where everything is virtualized. Access to hardware functionalities is very restricted and information about the hardware where jobs are running is difficult or impossible to gather.

Only the first case, HPC data centres with traditional HPC jobs, were already supported before HEROES. All the other use cases have been evaluated during the project and EAR has been extended to support them.

The last scenario is the only one with very limited functionality supported. In the project we are using AWS cloud instances where it is not possible to measure any performance counter, even at the process level, and no actions concerning power or CPU frequency can be performed. Currently we are working on a power model based on the application utilization to provide some hint on power consumption.

4 EAR extensions for HEROES

Three types of extensions have been implemented to support HEROES deployment and execution. First set of extensions are related to the deployment of the software. Before the project some options were manually set during the generation of Makefiles and configuration files. Second set of extensions are related to the initialization of the EARL. As already commented, before the project the EARL was designed to be executed in a fully deployed environment and was disabled automatically if it was not the case. The extensions implemented create a kind of dummy environment with required data types and settings to support EARL and it is executed with the existing features. Finally, the code itself of the EARL and auxiliary libraries have been extended to support limited systems where some (or all) of the required features are not available. Moreover, to be able to report EAR data in HPC scenarios with connectivity limitations, we have also implemented a new report plugin where data will be automatically gathered by the HEROES runtime.

4.1 Automatic deployment

The first extension was improving the automatic configuration so that it requires no input on the install phase, improving the detection of packages and creating alternatives when dependencies for certain functionalities are not found.

A specific compilation flag EARL_LIGHT has been created to disable the extra code to manage these limited scenarios to avoid any potential overhead in clusters where this is not required.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

4.2 Initialization

The *EAR* software is composed of many components, each of which has varying requirements and dependencies. One of the goals for *EAR* in HEROES is to be able to provide some of its functionalities even when some component is missing. For example, the *EAR Daemon*, which is the only component with superuser privileges, reads the memory bandwidth metric among other metrics. If the *EAR Daemon* is missing, this metric cannot be read, and therefore some backup should be put into place to run the rest of the program without any problem.

A full EAR installation consists of one *EARGM* per scheduler, one *EARDBD* per service node, one *EARD* per compute node, and one EARL per application. The *EARGM* is optional and is not required for the proper function of the software.

For cases where one or more components are missing, a more modular version of EAR has been implemented, called *EAR Light*. In that case, *EAR Light* loads alternative components to replace the capabilities of the missing features.

Every considered component has a “dummy” variant that is put in place in case everything else fails. This dummy component is a bare-bones alternative, way less potent than any other component for the task, but with the added bonus that it depends on no one, and therefore it should always be able to load.

In a complete installation of *EAR*, the execution sequence is the following (see Figure 4):
1. The plugin contacts with the *EAR Daemon*, notifying of a new job.
2. The *EARD* creates a shared memory region with this information.
3. The *EAR Library*, that boots later, as the executable is being run, reads this shared memory region, at first to see whether this shared region is his (by reading the job id and step id), and later when it needs information from the *EAR Daemon*.

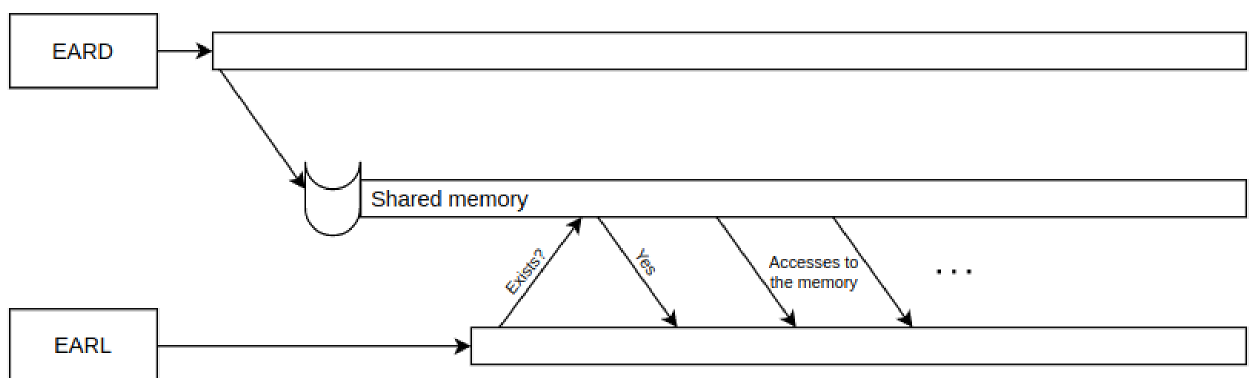

Figure 4 EARD/EARL interaction in a full installation

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

If, when reading the shared memory region, no file is found, or a file is found, but is not for this particular program, the *EAR Library* assumes that *EARD* has not been loaded properly, and will therefore load the dummy version (see Figure 5).

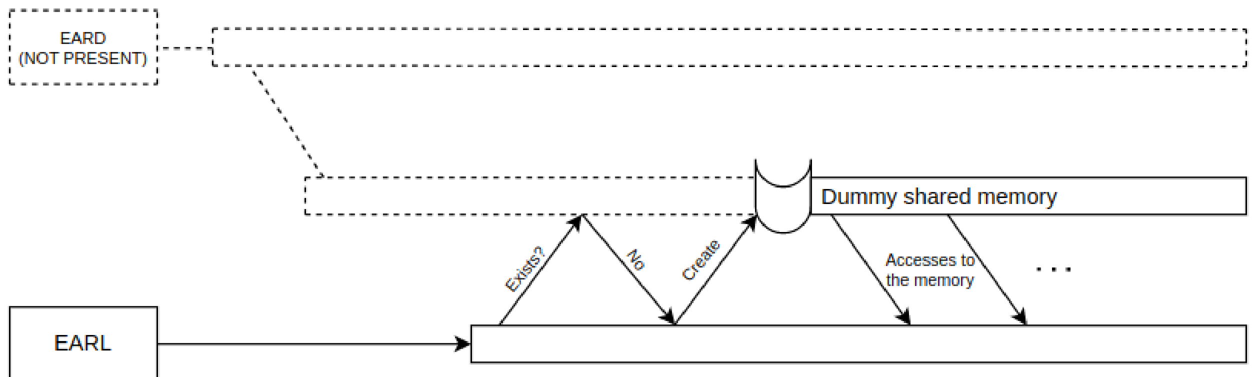

Figure 5 EARL behaviour in the absence of EARD

If the *EARD* is not detected, the following message pops up during execution:

```
[NODENAME]:EARD is not running
```

Where *[NODENAME]* is (quite self-explanatory) the node's name in which the program is running.

If the *EAR Daemon* is not found, the *EAR Library* initializes a shared region for the dummy EARD. It does so by creating the file *node_mgr_data* in *EAR_TMP*. The file has an *ear_njob* struct per possible CPU, which contains the job id, the step id, the node mask, and the creation time. This dummy region will contain the bare minimum so that the library's able to work even with the missing daemon component.

```
typedef struct ear_njob {
   job_id     jid;
   job_id     sid;
   cpu_set_t  node_mask;
   time_t     creation_time;
}
```

The presence or absence of components is detected automatically, and the corresponding modules are loaded depending on their prerequisites.

The following data structures have been considered, with each function creating a default configuration or a "fake" component:

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

- *Common fields in the cluster_conf. The function* tries to read the cluster configuration. If it can read it, it stores the cluster configuration into the dummy shared region, and if it cannot, it initializes the configuration paths with either the environment variables or with the default values for the paths, initializes a dummy island with nodes in them, energy tags with default settings, and the policies – being set by default to the monitoring policy.
- *EARL settings.* It initializes the settings for the EARL related to the EARD. It sets the user type to authorized, the policy in use to the default policy, the set frequency to the base (default) frequency, the p-state to the policy's p-state, the minimum and maximum power signature, the frequency of SIMD instructions, and the power cap to those of the node configuration, if present.
- *Node Manager info.* Initializes a node manager. The node manager includes information about other jobs sharing the same node.
- *List of CPU frequencies*
- *Application management information*. This data structure is used by the EARL to share (send) information to the EARD.
- *Powercap management*. Shared region setting the CPU and GPU mode to powercap mode.
- *Energy models coefficients*. Memory region with the coefficients used by the energy modes. Creates an empty coefficient shared area.

4.3 Support for execution in scenarios where EAR is not installed

Since some defaults are applied when configuration is missing and alternatives have been prepared for the cases where one or more dependencies are missing (be it EAR components or other software dependencies), EAR is ready to be executed even if only the EAR library is present (EAR not installed).

In environments where the hardware support is minimal, EAR works similarly to executions where EAR is not installed, dynamically detecting on startup which systems are available and deactivating the ones that cannot be used. Therefore, environments with minimum hardware functionalities (mostly cloud environments, but it also applies to highly virtualized environments) are supported with a reduced feature set which is entirely dependent on the availability of hardware functions.

EARL uses its own metrics and management libraries providing monitoring information and knobs for the CPU, Memory and GPU frequency settings. The API for these libraries has been extended to be completely transparent of the underlying hardware and in all the cases a dummy object has been provided. The API for all the classes includes a common set of functions and few specific depending on the particular metrics. Figure 6 shows the API used to measure FLOPS. The load function is the one including the intelligence to detect what is or not available. All the APIs look similar to this one with load, init, dispose, read, read_diff, etc functions. The fact the API is internally managing the low-level details makes the EARL more portable and agnostic of the different scenarios.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

```
state_t flops_load(topology_t *tp, int eard);
state_t flops_get_api(uint *api);
state_t flops_init(ctx_t *c);
state_t flops_dispose(ctx_t *c);
state_t flops_read(ctx_t *c, flops_t *fl);
// Data
state_t flops_read_diff(ctx_t *c, flops_t *fl2, flops_t *fl1, flops_t *flD, double *gfs);
state_t flops_read_copy(ctx_t *c, flops_t *fl2, flops_t *fl1, flops_t *flD, double *gfs);
state_t flops_data_diff(flops_t *fl2, flops_t *fl1, flops_t *flD, double *gfs);
/* Accumulates flops differences and return its data in GFLOPs (accepts NULL). */
state_t flops_data_accum(flops_t *fA, flops_t *fD, double *gfs);
state_t flops_data_copy(flops_t *dst, flops_t *src);
state_t flops_data_print(flops_t *flD, double gfs, int fd);
state_t flops_data_tostr(flops_t *flD, double gfs, char *buffer, size_t length);
```

Figure 6 FLOPS API example

4.4 HEROES report plugin

Currently, data centres do not allow users access from an external network unless it is done through secure shell connections. EAR uses a reporting mechanism based on plugins. Each component, including the EARL, can load 0..N report plugins to report to different targets. By default, the EARL reports to the EARD which in turns reports to the EARDBD and finally to the DB server. Even in the case EAR is fully deployed, the HEROES runtime cannot extract information directly from the DB because direct connections are not typically supported.
Figure 7 shows the report plugin API. Apart from the load operation, one function per EAR data type is included. The report plugin manager executes functions implemented by the plugin in the order they are loaded. It's not mandatory for a given plugin to implement all the functions. For example, the EARL only reports applications, loops, and events.

```
state_t report_load(const char *install_path, const char *libs);
state_t report_init(report_id_t *id, cluster_conf_t *cconf);
state_t report_dispose(report_id_t *id);
state_t report_applications(report_id_t *id, application_t *apps, uint count);
state_t report_loops(report_id_t *id, loop_t *loops, uint count);
state_t report_events(report_id_t *id, ear_event_t *eves, uint count);
state_t report_periodic_metrics(report_id_t *id, periodic_metric_t *mets, uint count);
state_t report_misc(report_id_t *id, uint type, const char *data, uint count);
```

Figure 7 Report plugin API

To circumvent the problem of remote access to accounting data, it was decided to make a new HEROES report plugin to store the job information using CSV files. The way to specify at runtime a new report plugin is 100% compatible with HEROES deployment requirements since it is done using an environment variable (SLURM_EAR_REPORT_ADD for SLURM systems).

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

These CSV files contain application information such as the power, bandwidth, CPI, GFLOPS per Watt, time, frequency, IO bandwidth, and loop information, which is quite similar only with added information such as the number of iterations. The user can give the plugin parameters to specify the data to be retrieved with greater precision, such as determining the particular job id or step id for which you want the information, the application name, or the energy tag.

Without *EARD*, however, no additional loop information is available since to get information on the loops, you need the *EARD* component running.

For an application, the header is the following:

```
HEROES_WFID;HEROES_JOBID;HEROES_USERID;HEROES_PROJECTID;HEROES_ORGID;MAX_POWER_W;NODENAME;JOBID;STEPID;USERID;GROUPID;JOBNAME;USER_ACC;ENERGY_TAG;POLICY;POLICY_TH;AVG_CPUFREQ_KHZ;AVG_IMCFREQ_KHZ;DEF_FREQ_KHZ;TIME_SEC;CPI;TPI;MEM_GBS;IO_MBS;PERC_MPI;DC_NODE_POWER_W;DRAM_POWER_W;PCK_POWER_W;CYCLES;INSTRUCTIONS;GFLOPS;L1_MISSES;L2_MISSES;L3_MISSES;SPOPS_SINGLE;SPOPS_128;SPOPS_256;SPOPS_512;DPOPS_SINGLE;DPOPS_128;DPOPS_256;DPOPS_512
```

For loops, the header looks like this:

```
HEROES_WFID;HEROES_JOBID;HEROES_USERID;HEROES_PROJECTID;HEROES_ORGID;JOBID;STEPID;NODENAME;AVG_CPUFREQ_KHZ;AVG_IMCFREQ_KHZ;DEF_FREQ_KHZ;ITER_TIME_SEC;CPI;TPI;MEM_GBS;IO_MBS;PERC_MPI;DC_NODE_POWER_W;DRAM_POWER_W;PCK_POWER_W;CYCLES;INSTRUCTIONS;GFLOPS;L1_MISSES;L2_MISSES;L3_MISSES;SPOPS_SINGLE;SPOPS_128;SPOPS_256;SPOPS_512;DPOPS_SINGLE;DPOPS_128;DPOPS_256;DPOPS_512;LOOPID;LOOP_NEST_LEVEL;LOOP_SIZE;ITERATIONS;TIMESTAMP
```

To turn on the report plugin, one must add the following option either to SLURM and the environment variable presented before:

```
--ear-user-db=[file]
```

An example output – for applications and loops, respectively – would look something like this:

```
HEROES_WFID;HEROES_JOBID;HEROES_USERID;HEROES_PROJECTID;HEROES_ORGID;MAX_POWER_W;NODENAME;JOBID;STEPID;USERID;GROUPID;JOBNAME;USER_ACC;ENERGY_TAG;POLICY;POLICY_TH;AVG_CPUFREQ_KHZ;AVG_IMCFREQ_KHZ;DEF_FREQ_KHZ;TIME_SEC;CPI;TPI;MEM_GBS;IO_MBS;PERC_MPI;DC_NODE_POWER_W;DRAM_POWER_W;PCK_POWER_W;CYCLES;INSTRUCTIONS;GFLOPS;L1_MISSES;L2_MISSES;L3_MISSES;SPOPS_SINGLE;SPOPS_128;SPOPS_256;SPOPS_512;DPOPS_SINGLE;DPOPS_128;DPOPS_256;DPOPS_512
NO_WRF_ID;NO_HER_ID;NO_USR_ID;NO_PRJ_ID;NO_ORG_ID;264.85;node;196705;0;username;                    ;script;acc;
;monitoring;0.000000;2375500;2393547;2400000;95.051268;0.379903;2.584720;15.810803;0.026875;4.373057;258.467999;14.366986;209.507982;5220193216645;13740859346246;74.629827;162564271308;4429764251;3334096733;285;1168;0;0;3373342475768;3070526496276;0;0
```

```
HEROES_WFID;HEROES_JOBID;HEROES_USERID;HEROES_PROJECTID;HEROES_ORGID;JOBID;STEPID;NODENAME;AVG_CPUFREQ_KHZ;AVG_IMCFREQ_KHZ;DEF_FREQ_KHZ;ITER_TIME_SEC;CPI;TPI;MEM_GBS;IO_MBS;PERC_MPI;DC_NODE_POWER_W;DRAM_POWER_W;PCK_POWER_W;CYCLES;INSTRUCTIONS;GFLOPS;L1_MISSES;L2_MISSES;L3_MISSES;SPOPS_SINGLE;SPOPS_128;SPOPS_256;SPOPS_512;DPOPS_SINGLE;DPOPS_128;DPOPS_256;DPOPS_512;LOOPID;LOOP_NEST_LEVEL;LOOP_SIZE;ITERATIONS;TIMESTAMP
NO_WRF_ID;NO_HER_ID;NO_USR_ID;NO_PRJ_ID;NO_ORG_ID;196705;0;node;2380500;2394349;2400000;0.160071;0.387044;3.276525;16.132765;0.013060;7.045602;262.332607;14.436775;212.745177;21002751421;54264563955;3.098753;665941331;17826984;13482102;0;0;0;0;17786415254;16364941894;0;0;38857729;0;51;66;11
NO_WRF_ID;NO_HER_ID;NO_USR_ID;NO_PRJ_ID;NO_ORG_ID;196705;0;node;2391000;2394418;2400000;0.158953;0.386004;5.527553;15.999811;0.018904;5.048738;261.484242;14.363370;209.795706;23501478119;60884089801;3.126855;764499914;20767734;17363155;113;0;0;0;34277744796;31492527892;0;0;38857729;0;51;135;22
NO_WRF_ID;NO_HER_ID;NO_USR_ID;NO_PRJ_ID;NO_ORG_ID;196705;0;node;2398000;2394509;2400000;0.170604;0.383573;8.101056;15.945289;0.019166;4.033125;264.847296;14.395778;209.183731;23350176370;60875509987;3.148762;759141431;20475533;17048655;113;0;0;0;50788914721;46595988030;0;0;38857729;0;51;200;32
NO_WRF_ID;NO_HER_ID;NO_USR_ID;NO_PRJ_ID;NO_ORG_ID;196705;0;node;2391000;2394394;2400000;0.173439;0.383306;10.668216;15.908038;0.018149;3.958848;240.646451;14.390808;208.981014;23326975842;60857302674;3.159287;742533299;18473165;15183824;117;0;0;0;67221370514;61715446924;0;0;38857729;0;51;261;41
NO_WRF_ID;NO_HER_ID;NO_USR_ID;NO_PRJ_ID;NO_ORG_ID;196705;0;node;2395500;2394364;2400000;0.156758;0.383440;5.224117;15.831580;0.004636;3.869720;259.440119;14.362150;209.172401;93324320835;243386922091;3.178104;3025691651;78376687;64380347;120;0;0;0;133217126440;122025909136;0;0;38857729;0;51;519;81
```

Table 1 and Table 2 show the list of the headers reported for applications and loops. Three groups of fields are reported: ID fields, for example HEROES_WFID or JOBID, performance metrics such as CPI and power metrics such as DC_NODE_POWER_W.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

Table 1 Application header

HEROES_WFID	For now they are text fields. There fields can change in the remaining months but they refer to specific HEROES IDs.
HEROES_JOBID	
HEROES_USERID	
HEROES_PROJECTID	
HEROES_ORGID	
MAX_POWER_W	The maximum power, in watts (W).
NODENAME	The name of the node the job is running in.
JOBID	The ID of the job generated by the scheduler.
STEPID	The ID of the particular step of the job generated by the scheduler.
USERID	The username of the user that submitted the job.
GROUPID	The group name of the user that submitted the job.
JOBNAME	The name of the submitted job.
USER_ACC	SLURM's user account.
ENERGY_TAG	The energy tag of the task running. Indicates if the program is CPU/Memory intensive, and similar things.
POLICY	The energy policy requested to EAR (monitoring, min_energy, min_time…).
POLICY_TH	Policy threshold.
AVG_CPUFREQ_KHZ	Average CPU frequency in kilohertz (kHz), includes all the cores.
AVG_IMCFREQ_KHZ	Average memory frequency in kilohertz (kHz), includes all the sockets.
DEF_FREQ_KHZ	Default CPU frequency in kilohertz (kHz). It can be different from the average depending on the type of instructions and the utilization.
TIME_SEC	Time in seconds (s).
CPI	Cycles per instructions. Low CPI means CPU bound. Close to 0,5 is low CPI. More than 1 is high.
TPI	Transactions to main memory per instruction. Is used by energy models and represents how memory bound the application is.
MEM_GBS	Main memory bandwith in gigabytes per second (GB/s). From experience, less than 10 is very low. Between 10- 50 or 60 GB/s is medium, more than this is high, more than 100 is very high.
IO_MBS	I/O (read & write) in megabytes per second (MB/sec).
PERC_MPI	Percentage of MPI time vs iteration time. It gives a hint about how efficient the application is in terms of load distribution and in communications.
DC_NODE_POWER_W	DC node power in watts (W), includes GPU power.
DRAM_POWER_W	DRAM power in watts (W).
PCK_POWER_W	Package (CPU+Cache) power in watts (W).
CYCLES	Number of CPU cycles and number of assembly instructions, respectively. Accumulated for all the processes in the node for the application. Cycles / Instructions = CPI
INSTRUCTIONS	
GFLOPS	Number of floating point operations divided by time in seconds. It's as indicator of how CPU intensive is the application.
L1_MISSES	Number of cache misses for every cache level.
L2_MISSES	
L3_MISSES	
SPOPS_SINGLE	Number of floating point operations for each type.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

SPOPS_128	SPOPS – Single precision operations (floats)
SPOPS_256	DPOPS – Double precision operations (doubles)
SPOPS_512	
DPOPS_SINGLE	
DPOPS_128	
DPOPS_256	
DPOPS_512	

The loop signature has some exclusive headers, not present in the application header:

Table 2 Exclusive headers for loops

"ITER_TIME_SEC"	The iteration time of the loop, in seconds (s).
"LOOPID"	The loop identifier.
"LOOP_NEST_LEVEL"	The nested level of the loop.
"LOOP_SIZE"	The size of the loop.
"ITERATION"	The number of iterations in that loop.
"TIMESTAMP"	The timestamp of the loop.

4.5 Singularity

Singularity is a widely used open-source virtualization and containerization tool. It is very popular for HPC contexts. It allows for greater reproducibility, making the programs less dependent on the environment they're being run on.

An example singularity command could look something like this:

```
singularity exec $IMAGE program
```

Where *IMAGE* is an environment variable that contains the path for the Singularity container, and *program* is the executable to be run. The work done concerning Singularity has been the evaluation of some use cases with EAR to identify potential extensions and new requirements. The evaluation showed us that there was no need to have new extensions and the only requirement to do the right configuration at deploy time was detecting if EAR was already installed on the system or not and set the environment variables to make EAR installation "visible" to the Singularity container. Because of that we will just include some examples of utilization.

Singularity also allows to add elements from outside of the image to be accessible from inside via environment variables. In particular, *SINGULARITY_BIND* is a comma separated string of pairs of paths *[path1]:[path2]*, such that *[path1]* in local will be mapped into *[path2]* in the image.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

To make *EAR* work, the following paths should be added to *SINGULARITY_BIND*: $EAR_INSTALL_PATH, $EAR_INSTALL_PATH/bin, $EAR_INSTALL_PATH/lib and $EAR_TMP.

Once the paths are deployed, to execute an OpenMPI application with Singularity and EARL it's only needed to do something similar to that example:

```
mpirun -np $N singularity exec $IMAGE erun --ear=on --program=program
```

A more complete example would look something like this:

```
export MPI_ROOT=/hpc/opt/openmpi4.1.1
export EXP_ROOT=[script path]
export PATH=$MPI_ROOT/bin:$PATH
export LD_LIBRARY_PATH=$MPI_ROOT:$LD_LIBRARY_PATH
export IMAGE=$EXP_ROOT/ubuntu_ompi.sif
export SINGULARITY_TMPDIR=$EXP_ROOT
export SINGULARITY_BIND=""
append SINGULARITY_BIND $EAR_INSTALL_PATH:$EAR_INSTALL_PATH:ro
append SINGULARITY_BIND $EAR_INSTALL_PATH/bin:$EAR_INSTALL_PATH/bin:ro
append SINGULARITY_BIND $EAR_INSTALL_PATH/lib:$EAR_INSTALL_PATH/lib:ro
append SINGULARITY_BIND $EAR_TMP:$EAR_TMP:rw
append SINGULARITY_BIND $MPI_ROOT/lib:$MPI_ROOT/lib:ro
append SINGULARITY_BIND $MPI_ROOT/bin:$MPI_ROOT/bin:ro
append SINGULARITY_BIND $EXP_ROOT:$EXP_ROOT:rw
append SINGULARITY_BIND /lib64:/lib64

export SINGULARITYENV_MPI_ROOT=$MPI_ROOT
export SINGULARITYENV_SLURM_EAR_REPORT_ADD=heroes.so
export SINGULARITYENV_APPEND_PATH=$MPI_ROOT/bin:$EAR_INSTALL_PATH/bin
export TMPDIR=$EXP_ROOT
export SINGULARITYENV_LD_LIBRARY_PATH=/lib64:$LD_LIBRARY_PATH
mpirun -np 24 singularity exec $IMAGE erun --ear=on --program=$EXP_ROOT/mpibinary
```

Where SINGULARITY_BIND and SINGULARITY_ENV are Singularity specific environment variables to bind paths and to define environment variables. We have executed some

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

preliminary experiments to evaluate the potential overhead when using Singularity. Even though it was not a complete evaluation, we observed the different in execution time was measured in the creation of the image and not during the execution of the applications.

4.6 Cloud

For the EARL support in cloud scenarios, we have been extending EAR in the same topics commented before about configuration, initialization etc. The main complexity is the fact in some could instances the is no information available at all. For example, there are no cpufreq drivers (the OS driver used for CPU frequency management), no performance counters etc. In this case the configuration of EARL is done based on OS information and some hardcoded information such as the TDP of the CPU types which is publicly available.

In this context the ongoing work consists of implementing some power estimation based on the available information about system (and application) utilization.

5 Summary

This deliverable describes the analysis of the requirements for the EAR software when being executed in all the HEROES scenarios. EAR has been extended to provide a better configurability during the HEROES deployment process and during the application execution. The goal is to support any scenario even when EAR is not fully installed or when being executed in a virtualized environment.

For this purpose, the initialization phase has been improved to automatically detect the system settings using OS information, which is always available, the APIs have been improved to be automatically configurable and always include a dummy version of classes, and a new report plugin has been created to be able to get the application metrics using standard protocols such as ssh.

Until the end of the project two main topics are pending. First, we will extend the data reported by EAR to include more semantics on the internals of the application to improve the resource selection done by the decision module. The EARL classifies the application at runtime as a previous phase for the energy policy execution. This classification can be potentially done with the metrics reported but it will help to accelerate the decision module strategies if the classification is already available. Second, it's still pending to improve the power model to provide a better power estimation for limited scenarios.

6 References

[1] J. Corbalan, L. Alonso, J. Aneas and L. Brochard, "Energy Optimization and Analysis with EAR," 2020 IEEE International Conference on Cluster Computing (CLUSTER), 2020, pp. 464-472, doi: 10.1109/CLUSTER49012.2020.00067.

[2] J. Corbalan, O. Vidal, L. Alonso and J. Aneas, "Explicit uncore frequency scaling for energy optimisation policies with EAR in Intel architectures," *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 572-581, doi: 10.1109/Cluster48925.2021.00089.

[3] EAR gitlab. https://gitlab.bsc.es/ear_team/ear

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

[4] EAR documentation. https://gitlab.bsc.es/ear_team/ear/-/wikis/home

[5] SLURM SPANK plugin documentation https://slurm.schedmd.com/spank.html

[6] Singularity web. https://sylabs.io/docs/

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.