Modular Workload Format: extending SWF for modular systems*

Julita Corbalan[1,2][0000−0002−3926−5634] and Marco D'Amico[2][0000−0002−6195−6204]

[1] Universitat Politecnica de Catalunya, Barcelona, Spain
[2] Barcelona Supercomputing Center (BSC), Barcelona, Spain
julita.corbalan@bsc.es, marcodamico@protonmail.com

Abstract. This paper presents the Modular Workload Format(MWF), a proposal for extending the widely accepted Standard Workload Format (SWF) for job scheduling evaluation. David Talby and Dror Feitelson proposed the SWF in 1999, allowing to describe data center workload in a synthesized way. Its simplicity, representing each job by a single line in a text file and including details to make job scheduling evaluation quite accurate, was part of its success. Using these years' experience but considering new system and workload characteristics, we propose an extension to support multiple steps in a single job, heterogeneous jobs, and relevant inputs not covered by the SWF as energy/power references. The goal of this contribution is to adapt the SWF to current trends in architectures and workloads. Moreover, we propose a simple approach for converting any already existing SWF trace file into an MWF trace file to be able to reuse already existing traces.

Keywords: Workload traces · Job scheduling evaluation

1 Introduction and motivation

Standard Workload Format (SWF) [9] is a widely accepted format in job scheduling research as a standard way to evaluate job scheduling policies. A repository of SWF traces provided by many HPC centers can be found at [8], and many research papers are using this format. There are also several proposals of workload models that generate job scheduling logs in SWF.

In the context of the DEEP-EST European project [1], we found some limitations when designing the job scheduling simulation methodology. Some workload characteristics, such as modular jobs, and some job features, such as power/energy data, were not considered, given they were not so relevant at the moment the SFW was proposed. A Modular System Architecture (MSA) [1] integrates compute modules (or sub-clusters) with different performance characteristics into a single heterogeneous system. Each module is a parallel, clustered

* This work is partially supported from the European Union's Horizon 2020 under grant agreement No. 754304 (DEEP-EST Project) and the Spanish grant PID2019-107255GB-C21

system of potentially large size. A federated network connects the module-specific interconnects. A module differs from a partition because it is a hardware organization, whereas a partition is a software concept. Of course, nothing prevents describing a module as a partition, but we want to consider the scenario where even in MSA there is a single submission point, there could potentially be many sub-schedulers dealing with the same partition names referring to different configurations. This option already exists in some schedulers such as SLURM with the `--clusters` option, and we propose to have both options, partition, and module (or cluster), for flexibility.

A modular job in this context is a job with multiple sub-components, each running in different modules simultaneously. In this work, we will refer as *jobs* to a request sent to the scheduler that can be a traditional HPC job with a simple mapping, i.e., one-request-one-application, or something more complex such as one allocation including several sub-components, with or without internal dependencies, executed in the same or different modules. Similar use cases can be found in other systems, such as systems running Slurm that supports heterogeneous jobs, or jobs with multiple internal executions, steps executed with *srun*, or even MPI jobs executing different binaries as a single application part of the same allocation. The concept of dependency is already present in the SWF, but we propose to go one step forward and incorporate types of dependencies and dynamic dependencies. Finally, we propose incorporating in the trace files a list of runtime events associated with each component (not mandatory) to introduce actions to be passed to simulators to take into account when simulating or analyzing workload traces.

We decided to analyze the SWF in detail and extend it with a proposal as much compatible with SWF as possible for traces reusability and with the same philosophy, a text file with a fixed number of columns, with each row describing a *job component*, the minimum unit of execution in our proposal.

In our scenario, one job can be internally composed of several components, each consuming a piece of the allocation or the whole but using resources sequentially or in a non-exclusive way. Moreover, each component characteristic could be modeled with a high precision starting from collected related metrics such as the Cycles per Instruction (CPI) and the bandwidth. This kind of information is very valuable for complex scenarios where runtime decisions are also simulated and not only the initial resource allocation. However, given this information is not provided by users at submission time and most of the schedulers do not automatically collect them, we avoided including all these metrics in the proposal to keep the number of columns to a reasonable value. However, it is advisable to include additional data related to components in separate files that allow more precise modeling. The additional data could be linked to MWF data using the component ID as an identifier.

MWF is compatible with SWF. SWF traces can be directly re-used by simply considering that each job in the SWF is one MWF job with a single component. All the extra fields in the MWF can be set to null values if the information is not available.

In the rest of this document, Section 2 analyses the SWF in deep and proposes the granularity of the fields, either job or component. Based on this analysis, Section 4 presents our proposal for MWF. To make our proposal more comprehensive, Section 6 presents, in a simplified way, few examples showing how the MWF can be used to represent different scenarios and some experiments already done using this format. Finally, Section 7 presents some conclusions and lessons learned.

2 Standard Workload Format

The SWF was defined in order to ease the use of workload logs and models. SWF allows simple workload analysis and or system's job scheduling simulation since they only need to parse a common standard format applied to multiple workloads. The SWF files are portable and easy to parse:

- Each workload is stored in a single ASCII file.
- Each job is represented by a single line in the file.
- Lines contain a predefined number of fields, which are mostly integers, separated by whitespace(s).
- Fields that are irrelevant for a specific log or model appear with a value of -1.
- Comments are allowed, and identified by lines that start with a ';'. In particular, files are expected to start with a set of header comments that define the environment or model.
- The same format is used for logs and model outputs.
- The format is completely defined, with no scope for user extensibility.

This last point is what we would like to reconsider after many years using the same format.

Current fields in SWF We have analysed fields to identify which ones are job-specific (J) or potentially component-specific (C). Those fields referring components will be replicated the MWF. Fields marked as (W) are used to specify job dependencies, i.e., workflows. [3]

1. (J) Job Number – a counter field, starting from one.
2. (J) Submit Time – in seconds. The earliest submit time in the log is zero, and usually, it is the submit time of the first job. The lines in the log are sorted by ascending submit times. It makes sense for jobs to also be numbered in this order.
3. (J/C) Wait Time – in seconds. The difference between the job's submit time and the time at which it actually started its execution. It is only relevant to real logs, not to models.

[3] The fields description comes from the SWF web page.

4. (J/C) Run Time – in seconds. The total execution time of the job, i.e., end time minus start time. We decided to use "wait time" and "run time" instead of the equivalent "start time" and "end time" because they are directly attributable to the scheduler and application, and are more suitable for models where only the run time is relevant. Note that when values are rounded to an integral number of seconds (as often happens in logs) a run time of 0 is possible and means the job ran for less than 0.5 seconds. On the other hand it is permissible to use floating point values for time fields.
5. (C) Number of Allocated Processors – an integer. In most cases this is also the number of processors the job uses; if the job does not use all of them, we typically don't know about it.
6. (C) Average CPU Time Used – both user and system, in seconds. This is the average over all processors of the CPU time used, and may therefore be smaller than the wall clock runtime. If a log contains the total CPU time used by all the processors, it is divided by the number of allocated processors to derive the average.
7. (C) Used Memory – in kilobytes. This is again the average per processor.
8. (C) Requested Number of Processors.
9. (C) Requested Time. This can be either runtime (measured in wallclock seconds), or average CPU time per processor (also in seconds) – the exact meaning is determined by a header comment. In many logs this field is used for the user runtime estimate (or upper bound) used in backfilling. If a log contains a request for total CPU time, it is divided by the number of requested processors.
10. (C) Requested Memory (again kilobytes per processor).
11. (J) Status 1 if the job was completed, 0 if it failed, and 5 if cancelled. If information about checkpointing or swapping is included, other values are also possible.
12. (J) User ID – a natural number, between one and the number of different users.
13. (J) Group ID – a natural number, between one and the number of different groups. Some systems control resource usage by groups rather than by individual users.
14. (J/C) Executable (Application) Number – a natural number, between one and the number of different applications appearing in the workload. in some logs, this might represent a script file used to run jobs rather than the executable directly; this should be noted in a header comment.
15. (J/C) Queue Number – a natural number, between one and the number of different queues in the system. The nature of the system's queues should be explained in a header comment. This field is where batch and interactive jobs should be differentiated: we suggest the convention of denoting interactive jobs by 0.
16. (J/C) Partition Number – a natural number, between one and the number of different partitions in the systems. The nature of the system's partitions should be explained in a header comment. For example, it is possible to use partition numbers to identify which machine in a cluster was used.

17. (W) Preceding Job Number – this is the number of a previous job in the workload, such that the current job can only start after the termination of this preceding job.
18. (W) Think Time from Preceding Job – this is the number of seconds that should elapse between the termination of the preceding job and the submission of this one.

3 Heterogeneous systems requirements

Modular architectures term comes from the DEEP-EST projects and stands for multiple clusters, or modules, with different specialized architectures working together as a single homogeneous system. The system software hides the complexity existing in this proposal. Even though our motivation comes from this complex scenario, heterogeneous systems already exist in many data centers as a simplified version of this use case. As an example, it is a common strategy in the current data center to have some computational nodes with specific characteristics such as extra memory of GPUs.

The SWF was designed in a context where jobs were considered as a whole and systems with multiple clusters were not usual. In recent years, having multiple clusters in a single data center became a typical scenario, but in many cases, the system workload was a set of disjoint workloads given the architecture differences. The only flexibility in terms of submission was the possibility to specify multiple partitions but always referring to the same HW characteristics.

With potentially highly specialized modules, or sub-clusters, this new hardware context opens to the possibility of considering new job profiles and use cases to be evaluated:

- Jobs asking for resources in a specific module.
- Jobs asking for resources in more than one module at the same time.
- Jobs asking for resources in more than one module but not at the same time, i.e., asking for additional resources dynamically.
- Jobs asking for not only the classical computing resources, e.g. GPUs, memory devices.
- Jobs with dependencies among them, i.e., workflows.

4 Modular Workload Format (MWF) proposal

We define a modular job as a scheduling unit belonging to a single user containing a single or multiple binaries. At submission time, the job will include an N ($N \geq 1$) list of requirements for allocation and components submission. All these allocations will be validated before any of these components start. These allocations could refer to different modules. Some components can be specified with different submission times representing the case where job allocation is dynamically increased.

The MWF includes the resource requirements and the resources allocated as it was proposed in SWF. Requirements represent the job scheduler's input, while resource allocation fields will be used for its output.

Each line will describe one component from one modular job. Dependencies can be specified between components of different modular jobs or the same job. Each workload trace file can include a system description, e.g., the available modules, the resources at each module, in the same way SFW did in the headers. The Component_Job_Id is a unique number and can be seen as the job_id in the SWF. The N potential components being part of the same job will share the same Modular_Job_Id.

Next subsections include the list of fields and a brief description of the semantic and valid values following a similar approach as in the SWF.

4.1 Modular Workload Format fields

Modular fields

1. **Modular_Job_Id** – An ID common to all the components of the modular job.
2. **Total_Components** – Number of components in the modular job (minimum one)
3. **Modular_Job_Name** – Text . Max of 16 chars. Job names is an user provided input and there can be more than one job name for the same executable.
4. **Submit_Modular_Job_Time** – in seconds. Submission time for the first set of components
5. **Wait_Modular_Job_Time** – in seconds. The difference between the job's submit time and the time at which it actually began to run (some of its components). It is not needed for evaluation, only for comparison between results.
6. **Modular_Requested_Time** – in seconds. Limit for the modular job. -1 if this value is not provided. In that case, the partition limit will be used
7. **Num_Components_At_Submit_Time** – Integer. This field is the number of components submitted together at modular submit time
8. **User_ID** - Integer
9. **Group_ID** - Integer

Job Component fields

10. **Component_Job_Id: Modular_Job_Id+ Offset** – This JOB ID is unique. It goes from Modular_Job_Id to Modular_Job_Id+(Total_Components-1).
11. **Component_Job_Name** – text . Max of 16 chars. Components names is an user provided input and there can be more than one job name for the same executable.
12. **Component_Wait_Time** – in seconds. The difference between the job's submit time and the time at which it actually began to run. It is not needed for evaluation, only for comparison between results

13. **Component_Run_Time** – in seconds. Integral number of seconds.
14. **Status** – 0 means COMPLETED with success. Values different from 0 will represent errors.

Job Component resource requirements description The job scheduler receives job component requirements, applies the job scheduler and resource selection policy, and reports a set of resources allocated. Resource allocation is reported for comparison, but it is not part of the input. One component will run in a single module. If one job needs more than one module, one component per module will be specified.

15. **Executable_Number** – a natural number, between one and the number of different applications appearing in the workload. in some logs, this might represent a script file used to run jobs rather than the executable directly; this should be noted in a header comment
16. **Requested_Partition_Name** – Text with the partition name; NA, if no specific partition is requested
17. **Requested_Nodes** – an integer. Number of nodes requested
18. **Requested_Processes_Per_Node** – an integer
19. **Requested_Cores_Per_Process** – an integer
20. **Requested_Cores_Per_Node** – an integer
21. **Requested_GPUS_Per_Node** –an integer
22. **Requested_Memory_Per_Node** – In KB
23. **Requested_Freq** – Requested frequency in Gigahertz, Format is min[-max]
24. **Reference Power** – Input average power in Watts. Input by user or a power model.
25. **Extra_requirements** – A set of keywords, potentially with & or | special characters. These constraints must be specified in the different modules to simplify resource selection. For instance, based on sbatch manual [12] $intel\&gpu, intel|amd$. This field can be used as a wildcard field to cover those new cases that could appear in the future.
26. **Licenses** – a comma separated list of requested licenses. name[:how_many, name2:] . Default 1

Component resource allocation description One component will run in a single Module. If one job needs more than one module, one component per module will be specified.

27. **Component_Module_Id** – 0 - Number of Modules (One component will run in a single module). Module ID where this component is executed
28. **Partition_Name** – Text with the partition name selected
29. **Nodes** – Number of allocated nodes
30. **Processes_Per_Node** – an integer
31. **Cores_Per_Node** – an integer
32. **Memory_Per_Node** – In KB (0 if not requested)

33. **GPUS_Per_Node** – an integer
34. **Average_CPU_Time** – an integer
35. **Freq** – frequency in Gigahertz
36. **Average Power** – measured average power in Watts
37. **Other_resources**

Dependencies

38. **After_Component_Job_Id** – This component must start after job ID. -1 if there is no dependency
39. **Dependency_Type** – -1=NO DEP/0=DYNAMIC/1=AFTER/ 2=AFTERANY/3=AFTEROK/4=AFTERNOTOK//5=SINGLE.
    This list of types of dependencies is inspired by Slurm dependencies. DYNAMIC is an additional type defined here.
    - -1 means there is no dependency.
    - DYNAMIC means the component must be started N seconds after AFTER_COMPONENT_JOB_ID. The number of seconds is defined in the next field, and in that case it is relative to the dependent job start time.
    - AFTEROK/AFTERNOTOK – This job can begin execution after the specified component_id have successfully/not successfully executed.
    - SINGLE is which job can begin execution after any previously launched component_id by the same user and sharing the same component_id name have terminated.
40. **Component_Think_Time** – in seconds. When DYNAMIC is selected, it corresponds to the requested delay from the start of the first component to the start of this component. Otherwise, it is related to the job finalization overhead, like the SWF *Think Time from Preceding Job* field.

Component-level events

41. **Sched_event_list** – in seconds. It models events called by the job that impact the job scheduling. It is a list of comma-separated *key:value* elements, with the key representing the event type, an ID or keyword, and value the number of seconds passed from the start of the job until the event. Keys depends on the specific simulator and it's a way to specify runtime actions. -1 means no events. E.g.: "ChangeDepToAfter:650".

4.2 Headers

The SWF includes a header section with comments describing workload and system characteristics. These headers describe the architecture where the trace was collected and are interpreted as comments when reading the trace file. These headers are **not mandatory** and are included to characterize the system at which the trace file was recorded.

We include here only new proposals and not all the headers already proposed in the SWF. Since this section is optional, it is not needed to be as exhaustive as with SWF fields. Two new headers are proposed to support having N modules. For each module, a module number will be provided together with headers referring to cluster characteristics.

- **(new)NumberModules**: Number of modules in the system, for each Module
- **(new)ModuleNumber**: from 0 to max modules

Headers will include then a common section Version..EndTime, NumberModules, $[ModuleNumber, Computer...Partition]$ repeated N times.

5 From SWF to MWF

Adapting SWF trace files to the new proposal is as easy as associating a job in the SWF with a job with one component in the MWF. Given we have defined all the new fields as optional, except IDs, they can be easily defined with NULL values, while the other fields are mapped with the following rules:

- Job Number is mapped to Modular Job ID.
- Submit Time, Wait Time, Run Time, Requested time are mapped to their Modular respective.
- Number of allocated Processors is mapped to Nodes and Cores Per Node by dividing its value by the node's number of cores.
- Used Memory and Requested Memory are mapped to Requested_Memory_Per_Node and Memory_Per_Node by dividing the total amount by the number allocated of nodes.
- Status, User ID, Group ID, Average CPU Time Used, and Executable Number exist in both formats, Partition Number is mapped to Partition_Name.
- Preeceding Job Number is mapped to Dependency_Type of type 3, Think Time from Preceding Job to Component_Think_Time.
- Queue Number can be mapped on Extra_requirements, or integrated in the partition mechanism as many center nowadays do.

An parser example is available in the BSC Slurm Simulator Github repository [16].

6 MWF experiences

In this section, we present a list of use cases that can be represented using the MWF. We have included a subset of the fields to make it readable. The use cases focus on jobs with one or multiple components and some example with dependencies.

The main fields are presented in Table 1, showing the following use cases:

1. Jobs with a single step: a classical job submission is shown in job modular id 1.
2. Jobs with multiple steps: job 2, made up of two components, with id 2 and 3, represents a job with two sub-components. Component 3 starts after component 2 as shown in Figure 1, because of the AfterCompJobID parameter set to 2. Using Slurm as an example, it represents two steps of a jobs. If component 3 does not set AfterCompJobID it means the components run in parallel, as represented in Figure 2.

Fig. 1. Job with multiple steps.

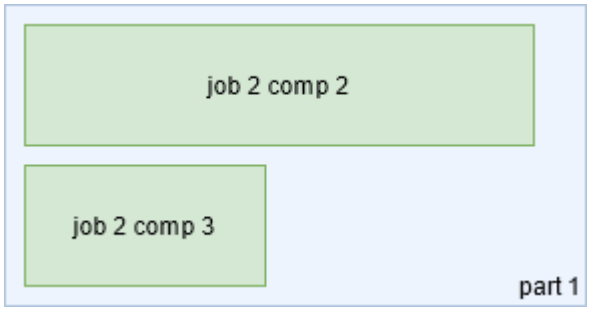

Fig. 2. Job with multiple parallel steps.

3. Jobs with multiple heterogeneous steps: similar to the example before, the steps request a different type of resources, for instance, different modules or partitions. In the case of job 12, also shown in Figure 3, the first job asks for *part1* and the second and third job ask for *part2*.
4. Jobs with dependencies (workflows): job 4, made up of components 4 and 5, is a workflow in which component 5 only runs after component 4 starts. Job 6, made up of three components, models a dynamic dependency in which the third component starts 300 seconds after the start of the first component, as in Figure 4. Finally, jobs 9, 10, and 11 in Figure 5 represent another workflow, in which each job starts after the previous one terminates. Note that in this case jobs are not scheduled as a single entity, like the case of jobs with multiple steps, so, depending on the scheduling, there can be large delays between the end of a component end the start of the next.

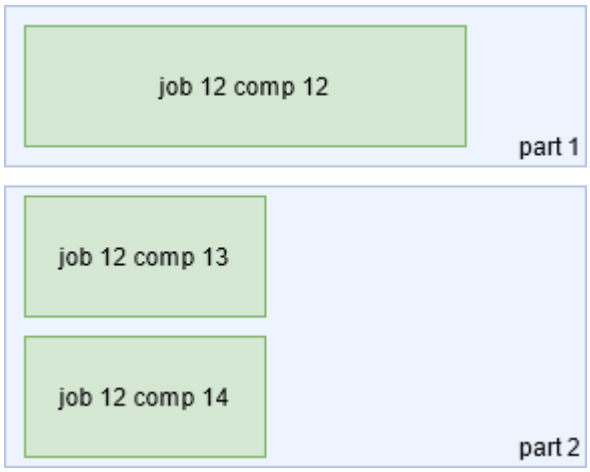

Fig. 3. Job with multiple heterogeneous steps.

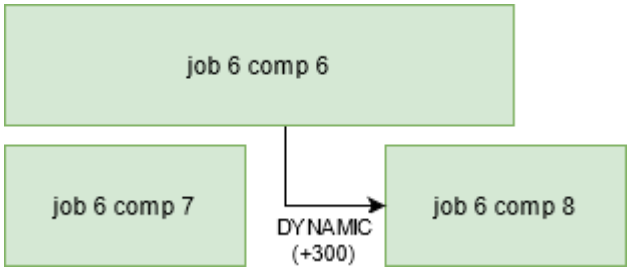

Fig. 4. Workflow with a dynamic dependency.

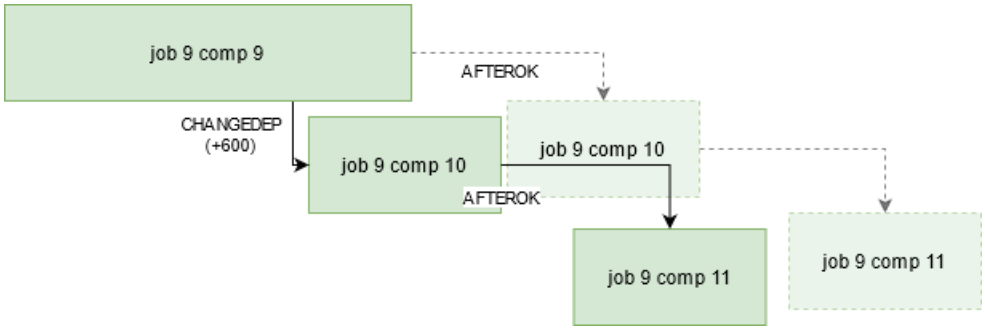

Fig. 5. Workflow using Sched_event_time to change dependency.

5. Jobs with events to be processed by the scheduler: in the workflow in Figure 5 made up of jobs 9, 10, and 11, job 10 uses Sched_event_time to run a change of its dependency, from AFTEROK to AFTER at time 600.
6. Job with extra resource requirements: jobs 11 and 12 request non CPU resources by specifying them in the Extra_Requirements field in the format "type:value" or only "type" if the values is 1. Multiple requirements can be connected with logical operations or regular expressions. In the example, jobs are asking for one Field Programmable Gate Array (FPGA).

For a better understanding of the example, we have used the text corresponding with the type of dependency rather than the number in column DepType and we have shortened the fields names.

Mod. JobID	Num Comp.	Mod. JobName	Comp. JobID	Exta. Requir.	Partition	AfterComp JobID	Dep Type	SchedEventList Time
1	1	job1	1	–	Default	-1	-1	-1
2	2	job2	2	–	Default	-1	-1	-1
2	2	job2	3	–	Default	2	-1	-1
4	2	job3	4	–	Default	-1	-1	-1
4	2	job3	5	–	Default	4	AFTER	-1
6	3	job4	6	–	Default	-1	-1	-1
6	3	job4	7	–	Default	-1	-1	-1
6	3	job4	8	–	Default	6	DYNAMIC	-1
9	1	job5	9	–	Default	-1	-1	-1
10	1	job6	10	–	Default	9	AFTEROK	changeDep:600
11	1	job7	11	*FPGA*	Default	10	AFTEROK	-1
12	2	job8	12	*FPGA*	part1	-1	-1	-1
12	2	job8	13	*FPGA*	part2	-1	-1	-1
12	2	job8	14	*FPGA*	part2	-1	-1	-1

Table 1. MWF example workload presenting main fields for different categories of jobs.

7 Conclusions

This paper is a proposal for considering the extension of SWF to be adapted to new systems and workloads. We call this proposal the Modular Workload Format (MWF) because it has been developed in the DEEP-EST project context, where the scheduler manages multiple modules, i.e., sub-clusters, as a single cluster. This kind of architecture is becoming popular since it allows users to ask for specialized resources. However, given their characteristics, it is impossible to express the semantics of our workloads with the already existing format.

To make it possible to use the existing traces and models, our proposal is a compatible format where existing traces could be migrated to the new format

with a straightforward approach. We propose maintaining the format as simple as possible, following the main criteria used when defining the SWF.

The main difference compared with the SWF is the possibility to define jobs with multiple components, each one with different requirements and potentially start times. We have also included additional fields to perform energy evaluations, and finally, some extra fields for demanding heterogeneous resources and dynamic workflow management.

This format has been used to evaluate job scheduling policies developed in the context of the DEEP-EST project. Workloads include job submission to multiple modules, new types of resource requirements specifics for some modules and components, and dynamic workflows. The current proposal is the result of the four years of experience after all these experiments.

References

1. DEEP-EST project http://www.deep-projects.eu/
2. DEEP-ER Deliverables, http://www.deep-projects.eu/project/deliverables.html
3. JUBE Online Documentation, https://apps.fz-juelich.de/jsc/jube/jube2/docu/index.html
4. MPI LinkTest, http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/LinkTest/_node.html
5. SIONlib, http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SIONlib/_node.html
6. IOR, https://github.com/hpc/ior
7. HDF5 Group, https://support.hdfgroup.org/HDF5/
8. The Standard Workload Format, http://www.cs.huji.ac.il/labs/parallel/workload/swf.html
9. Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby, *Benchmarks and Standards for the Evaluation of Parallel Job Schedulers*. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1999, Lect. Notes Comput. Sci. vol. 1659, pp. 66-89.
10. Morris A. Jette, Andy B. Yoo and Mark Grondona: *SLURM: Simple Linux Utility for Resource Management*, in Proceedings of the 9th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP), Springer, Lecture Notes in Computer Science (LNCS), volume 2862, pages 44–60, http://dx.doi.org/10.1007/10968987_3
11. SchedMD: *Slurm Workload Manager* [Online], https://slurm.schedmd.com/
12. sbatch: Submit a batch script to Slurm, https://slurm.schedmd.com/sbatch.html
13. *Heterogeneous Resources and MPMD*, https://slurm.schedmd.com/SLUG15/Heterogeneous_Resources_and_MPMD.pdf
14. SLURM: Heterogeneous job Support, https://slurm.schedmd.com/SLUG17/HeterogeneousJobs.pdf
15. DEEP-EST Deliverable 1.1 Application co-design input https://www.deep-projects.eu/images/materials/D11.pdf
16. BSC Slurm Simulator code, https://github.com/BSC-RM/slurm_simulator_tools