D3.3

Interim report on system middleware for workflows and resilience

Jan Jona Javoršek, Barbara Krašovec, and Alja Prah

Due date of deliverable: 31/12/2024 (month 24)
Actual submission date: 30/12/2024
Final version: 30/12/2024

Lead beneficiary: IJS (participant number 10)
Dissemination level: PU - Public

Document information

Project acronym:	MAX
Project full title:	Materials Design at the Exascale
Research Action Project type:	Centres of Excellence for HPC applications
EC Grant agreement no.:	101093374
Project starting / end date:	01/01/2023 (month 1) / 31/12/2026 (month 48)
Website:	www.max-centre.eu
Deliverable No.:	D 3.3

Authors: Jan Jona Javoršek, Barbara Krašovec, and Alja Prah

To be cited as: J.J. Javoršek, B. Krašovec, and A. Prah (2024): Interim report on system middleware for workflows and resilience. Deliverable D3.3 of the HORIZON-EUROHPC-JU-2021-COE-01 project MAX (final version as of 30/12/2024). EC grant agreement no: 101093374, IJS, Jožef Stefan Institute.

Disclaimer:

Versioning and contribution history:

Version	Date	Author	Note
0.1	10/12/2024	Jan Jona Javoršek	General draft
0.2	19/12/2024	Jan Jona Javošek, Alja Prah	Updates, graphics, additional material

0.3	20/12/2924	Alja Prah, Barbara Krašovec, Jan Javoršek	Additional material, conclusion, executive summary

D3.3 Interim report on system middleware for workflows and resilience

Contents

1 Executive Summary

It has become clear that among the basic requirements for successful scalability on exascale systems, HPC environments and the codes running in them have to work around the issues of increasing instability of large parallel tasks on very large bleeding edge systems. Increases of failure rates caused by larger number of components, more complex and longer workflows, interactions of disparate subsystems and complexities of modern modular architectures have to be offset with improvements not just in performance and scalability, but also in resilience and workflow management. Unmanaged failures have detrimental effects on machine utilizations, user satisfaction, general reliability, and time to result.
In order to ensure that MAX codes and deployments incorporate appropriate reliability and workflow management features, we have to consider this issue in development, testing and deployments. This document provides a report on several key areas where we analysed tools and developments that are most appropriate, within the current advanced ecosystem, to enhance the MaX codes and performance. These include: Analytics and performance tools (see Section 3) to be used to identify bottlenecks and performance issues, highlighting areas where scalability, performance, and efficiency could be improved. New features in schedulers for reliability and resilience support have been tested (see Section 4), with a particular emphasis on Slurm, and different workload managers and wordflow features of resource schedulers have been evaluated (see Section 5) to find the most effective solutions for our needs. Another key aspect under consideration was data handling (see Section 6 and 7), where we aimed to make the best use of available distributed storage solutions to achieve optimal performance.

Our study has generated recommendations for general purpose and specialized profilers and performance analysis tools used to improve performance, scalability and efficiency of running MaX codes on the EuroHPC machines. Tools for analysis of I/O performance, a common issue in scalability that requires understanding of a specific machine architecture, have also been included. Converged architectures and flexible workload solutions for modern converged machines will concentrate on Flux, Slurm, and Kubernetes, but we analysed specific plugins and solutions. Distributed storage and local fast storage solutions needed to support scalability of our workflows have been tested in different set-ups. Checkpoint and restart solutions that can

be adopted within recommended frameworks have been presented and analysed in preparation of an extensive study on different EuroHPC hosting entity architectures.
Our present report will form the basis of work with hosting entities and code developers in the next year as well as a point of departure for re-evaluation based on design decisions in the EuroHPC ecosystem and development of exascale workflows within MAX regarding the data requirements.

2 Mission Statement

Within the MAX project, the optimization and development of lighthouse codes – algorithms, applications and libraries, is a cornerstone, crucial for support of large communities and reflects on a non-negligeable load within the scientific use-cases of the European HPC infrastructure. Due to the rapid development of modern modular HPC architectures and the technology disruption brought in by exascale development, we envisage multiple new technologies will become relevant to the implementation of exascale workflows and optimization of exascale payloads. during the implementation of the project. This report is a mid-term accounting of the work of Task 3.3, dealing with understanding and evaluating new and emerging technologies that could become worth our consideration and potential immediate exploitation, especially from the point of view of scalability, optimization, resilience, and data exchange. This effort is important in order to keep our core code and library development not just current, but ready for the immediate future where we will be adapting it to new technologies and conditions in the state-of-the-art HPC environment of EuroHPC hosting entities. Since code development, adaptation and optimization takes considerable time, it is crucial to be able to include future trends in any decision that weighs on development priorities early enough.
This task analyses and evaluates (using experimental deployments, benchmark runs, comparative analyses and collaboration with administrator and HPC engineers) the use of advanced tools for resource management, scheduling, data handling, fast distributed storage, in-system data management, check-pointing, and resilience that are available or are being developed in other parts of the HPC ecosystem.
Re-stating the projected goals: among the technologies we expected to include in the analysis were the outcomes of the DEEP projects[1], next-generation schedulers such as Flux, storage

[1] https://www.deep-projects.eu/

solutions such as BeeGFS On Demand,[2] and other innovations in node-local storage, parallel I/O or distributed network disk image solutions, the use of object stores (i.e., S3 via CEPH[3,4]), and fast NoSQL databases such as Cassandra[5]. The task collaborates and coordinates with WP4 (T4.1 Co-design, technology exploitation & energy efficiency and T4.2 Exploitation of advanced hardware) to coordinate development.
The most promising technologies will drive some of the planned enhancements in the Tasks of WP2. Materials-science workflow needs will also help with the co-design of future middleware technologies, in cooperation with WP4, in particular on the modular supercomputing architecture (MSA). We plan to focus on benchmarking and practical experiments in the next year.

In the following we present a report on several key areas where e, within the group of task 3.3, we analysed tools and developments that can be used to enhance the MaX codes and performance. We have researched a number of useful analytics and performance tools (see Section 3) that can be used to help us identify bottlenecks and bugs in the code, highlighting areas where scalability, performance, and efficiency could be improved. Additionally, we tested an extensive number of new features in schedulers (see Section 4), with a particular emphasis on Slurm, and evaluated different workload managers (see Section 5) to find the most effective solutions for our needs. Another key aspect was data handling (see Section 6 and 7), where we aimed to make the best use of available distributed storage solutions to achieve optimal performance. This included running I/O profiling on the workloads to enhance data handling efficiency.

[2] BeeGFS is an implementation of a parallel filesystem. BeeGFS On Demand is a scheme to take advantage of fast local nodes to create a virtual file system for the duration of a process.

[3] CEPH is used for large storage as well as software and virtual machine supporting storage in a number of new systems as well as a distributed HPC storage solution. CEPH is being used as the main storage system in a number of new EuroHPC systems, including HPC Vega.

[4] K. Jeong, C. Duffy, J.-S. Kim, and J. Lee, in 2019 27th International Conference on Parallel, Distributed and Network-Based Processing (PDP) (2019), pp. 446–451

[5] https://cassandra.apache.org

3 Monitoring and Observability

3.1 Score-P

Score-P[6] is a community-maintained instrumentation and measurement infrastructure to collect performance data from HPC applications, developed by a consortium of partners. It is available as open-source under the 3-clause BSD license. Score-P is described as easy to use, highly scalable, and able to generate both summarising call-path profiles and detailed event traces.[7] Score-P provides the foundation for a number of well-established performance analysis tools since it uses open and well defined data formats: CUBE4[8] for profile data and the Open Trace Format 2 (OTF2)[9] for event traces, so that Score-P's event traces can be manually examined using trace visualisers or automatically analysed using the Scalasca Trace Tools (analysis and report in the next section). The captured call-path profiles can also be explored using dedicated tools, such as the Cube performance report explorer[10] as well as TAU[11] or be used to generate empirical performance models with Extra-P (see below).

Score-P mainly relies on instrumentation, i. e., the insertion of hooks into the application code that call into the Score-P run-time libraries at important points during the execution. There are a number of methods to add the instrumentation to an application without excessive modification of the source code, such as by using compiler flags, standardised interfaces such as PMPI, OpenMP, OpenCL and OpenACC tool interfaces, as well as the CUDA Profiling Tools Interface (CUPTI). Alternatively, one can use even more involved approaches, such as source-to-source translation, the Score-P instrumentation API to manually annotate the source

[6] A. Knüpfer et al. "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir". In: Tools for High Performance Computing 2011. Ed. by H. Brunst et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 79–91. DOI: 10.1007/978-3-642-31476-6_7

[7] Score-P website: https://www.score-p.org (Visited 2024-12-05)

[8] M. Geimer et al. "Further Improving the Scalability of the Scalasca Toolset". In: Applied Parallel and Scientific Computing. Ed. by K. Jónasson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 463–473. DOI: 10.1007/978-3-642-28145-7_45.

[9] D. Eschweiler et al. "Open Trace Format 2 – The Next Generation of Scalable Trace Formats and Support Libraries". In: Advances in Parallel Computing (Proc. of the Intl. Conference on Parallel Computing, ParCo) 22 (2012), pp. 481–490. DOI: 10.3233/978-1-61499-041-3-481.

[10] P. Saviankou et al. "Cube v4: From Performance Report Explorer to Performance Analysis Tool". In: Procedia Computer Science 51 (June 2015), pp. 1343–1352. DOI: 10.1016/j.procs. 2015.05.320.

[11] S. Shende,A. D. Malony. "The Tau Parallel Performance System". In: The International Journal of High Performance Computing Applications 20.2 (2006), pp. 287–311. DOI: 10.1177/ 1094342006064482

code, or an API instrumentation wrapper generator for third-party C/C++ libraries used by the application.

At run-time, Score-P is called by the instrumentation hooks that trigger callbacks in the measurement libraries. The system stores these events with the current high-resolution timestamps and event-specific data points (number of bytes transferred, hardware performance counter data via perf or Performance Application Programming Interface (PAPI)[12] etc.).

We found the use of Score-P quite challenging and expect most users would prefer to use eBPF-based instrumentation, measurement and observability tools where available. However, eBPF instrumentation of HPC workloads is still quite challenging: the challenges of observation of kernel-level events and the fact that scientific computing software, including MAX codes, often involves cross-language linking (C/Fortran) and tracing of non-code components.[13] Including additional tools (i.e. NUMACTL, taskset, vmtouch, and sysbench just for NUMA process placement) makes eBPF even more challenging. Currently, eBPF is not yet extensively used in HPC system performance engineering, and while it is believed to bring about a paradigm shift and gain an important role in future large-scale parallel software analysis architecture, currently Score-P is a welcome solution which provides HPC-centric solutions. Analysing application and library interplay as well as dynamically adjusting performance monitoring and profiling based on the specific characteristics of the workload, it is possible to reduce measurement overhead and provide quite accurate performance insights. While the need for instrumentation is a significant limitation, instrumentation on the level of libraries can often suffice for many observability, software analysis and optimization tasks. Availability of Scalasca Trace Tools and Extra-P makes the solution even more useful for our developers.

3.2 Scalasca Trace Tools

Scalasca Trace Tools[14,15,16] are a collection of trace-based performance analysis tools built on top of the Score-P instrumentation and measurement infrastructure introduced above. They are

[12] D. Terpstra et al. “Collecting Performance Data with PAPI-C”. In: Tools for High Performance Computing 2009. Ed. by M. S. Müller et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 157–173. DOI: doi.org/10.1007/978-3-642-11261-4_11.

[13] Wang, C., 2023. Performance Engineering on HPC Clusters.

[14] Scalasca website. URL: (visited on 09/23/2021)

[15] I. Zhukov et al. “Scalasca v2: Back to the Future”. In: Tools for High Performance Computing 2014. Ed. by C. Niethammer et al. Springer International Publishing, 2015, pp. 1–24. DOI: 10.1007/978-3-319-16012-2_1.

[16] DEEP-SEA Project Deliverables: D3.1 Software Specification: https://deep-projects.eu/wp-content/uploads/2023/09/DEEP-SEA_D3.1_Software-specification_Published.pdf

also available as open-source under the 3-clause BSD license and have been specifically designed for use on large-scale HPC systems. A distinctive feature of the Scalasca Trace Tools is its scalable automatic trace-analysis component, which provides the ability to identify wait states (off-cpu times) that occur due to unevenly distributed workloads or similar imbalances.[17] The trace analyser enables finding the root causes of waiting and estimate their impact,[18] as well as identify activities that are on the critical path of the target application and therefore constitute optimization-critical parts of the application.[19]
The analysis tools function as a parallel replay re-enacting the communication and synchronisation operations performed by the target application using similar operations to effectively exploit the memory and processing capabilities of the HPC system and generate an enriched call-path profile including additional higher-level metrics in CUBE4 format,[20] so that the standard tools for Cube/Extra-P can be used for analysis (i. e., Cube, TAU ParaProf/PerfExplorer, and Extra-P).

On the HPC Vega machine, where we performed our assessment, Score-P and Scalasca are available as EasyBuild modules. To collect measurements on any code, it must first be instrumented, i.e., additional code must be inserted into the original source code to collect performance data during its execution. This can either be done either manually or by using compiler wrappers. We automatically instrumented Quantum ESPRESSO and Siesta on Vega with Score-P and ran some tests in order to collect information about function calls, MPI events, synchronization events, communication events and so on. It is good practice to compare the instrumented run with a non-instrumented run to evaluate the overhead of Score-P instrumentation.

---

17 M. Geimer et al. “A scalable tool architecture for diagnosing wait states in massively parallel applications”. In: Parallel Computing 35.7 (July 2009), pp. 375–388. ISSN: 0167-8191. DOI: https://doi.org/10.1016/j.parco.2009.02.003

18 Böhme et al. “Identifying the Root Causes of Wait States in Large-Scale Parallel Applications”. In: ACM Trans. Parallel Comput. 3.2 (July 2016). ISSN: 2329-4949. DOI: 10.1145/2934661.

19 D. Böhme et al. “Scalable Critical-Path Based Performance Analysis”. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium. May 2012, pp. 1330–1340. DOI: 10.1109/IPDPS.2012.120.

20 CUBE4 Open call-path profile format used by Score-P, the Scalasca Trace Tools, and Extra-P.

```
vglogin0005: ~/max/AUSURF112/scorep_N2xn16xc4xnk2/> scorep-score -r profile.cubex

Estimated aggregate size of event trace:                   3237MB
Estimated requirements for largest trace buffer (max_buf): 303MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       305MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=305MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type  max_buf[B]     visits  time[s] time[%] time/visit[us]  region
        ALL 317,337,019 91,943,498 32804.62   100.0         356.79  ALL
        MPI 206,543,980 24,005,746  6602.20    20.1         275.03  MPI
        USR  98,262,658 59,874,504  9251.46    28.2         154.51  USR
        COM  13,167,128  8,063,232 16950.96    51.7        2102.25  COM
     SCOREP          71         16     0.00     0.0         137.30  SCOREP

        MPI 116,162,088 12,945,030  4494.27    13.7         347.18  MPI_Bcast
        MPI  82,621,428  9,228,364   176.48     0.5          19.12  MPI_Allreduce
        USR  24,315,200 14,963,200     2.75     0.0           0.18  wgauss
        USR  10,178,064  6,256,460     7.17     0.0           1.15  cclock
        USR  10,177,492  6,256,064    18.12     0.1           2.90  scnds
```

Figure 1: Scoring the measurement. Functions are divided into regions, where ALL stands for all functions, MPI for MPI functions, OMP for OpenMP constructs, COM for all functions that appear on a call path to any functions in other groups, USR for all user functions that are not in the COM group and SCOREP collects all functions related to the measurements themselves. Other groups include SHMEM, PTHREAD, CUDA, OPENCL, OPENACC and MEMORY groups.

A preliminary analysis of the results (i.e., the generated CUBE4 profile.cubex file, see Figure 1) via the CLI informs us of the number of different functions called in different regions, as well as the time spent in these functions and the average time per visit. We also get an estimate of the memory required for trace analysis. Some of the functions, which we believe cause significant overhead (due to short but numerous visit times), but are otherwise relatively insignificant, can be excluded from further analysis by employing a filter file (this allows us to lower the memory requirements for trace analysis significantly and reduce measurement overhead to an acceptable level). However, careful selection of the functions to filter is essential, as over-filtering can reduce the resolution of the data and obscure key performance hotspots.

When running trace analysis (via the scan keyword), some additional variables must be considered (e.g. SCOREP_FILTERING_FILE, SCOREP_TOTAL_MEMORY, SCOREP_ENABLE_TRACING). For more accurate results in an HPC environment, it is also advisable to set SCAN_ANALYZE_OPTS to “--time-correct”, which enables a correction algorithm that ensures that asynchrony between local clocks on individual compute nodes does not affect final results.

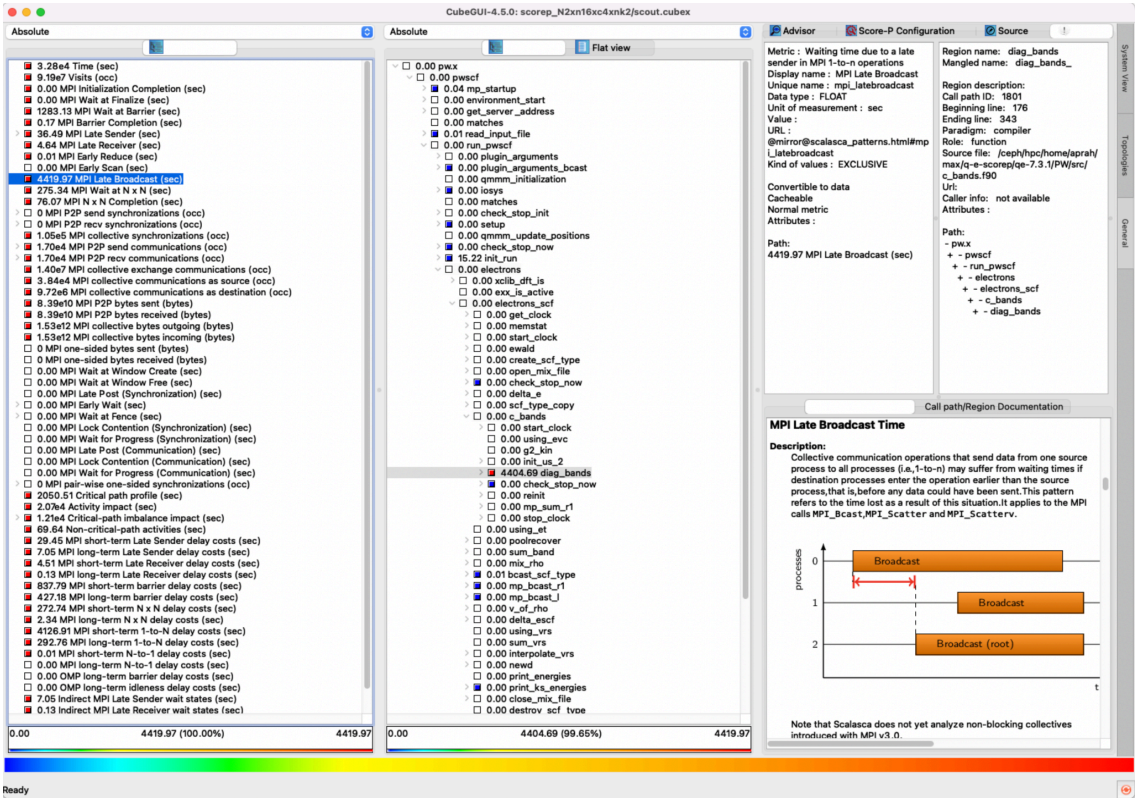

Figure 2: Using the CUBE GUI to analyze tracing data.

The obtained data can be visualized using the CUBE GUI (Figure 2), which is divided into three parts or dimensions – the metric tree, the call tree, and the system tree. All three can show inclusive values or can be further divided into showing exclusive values by expanding specific metrics. The metric tree (left) shows performance metrics such as time and the number of visits, the call tree (middle) shows all the call paths onto which metric values can be mapped and the system tree shows the metric values mapped onto specific parts of the system (machines, compute nodes, processes, threads). More information about specific metrics can be obtained by looking at the documentation for each specific metric (lower right).

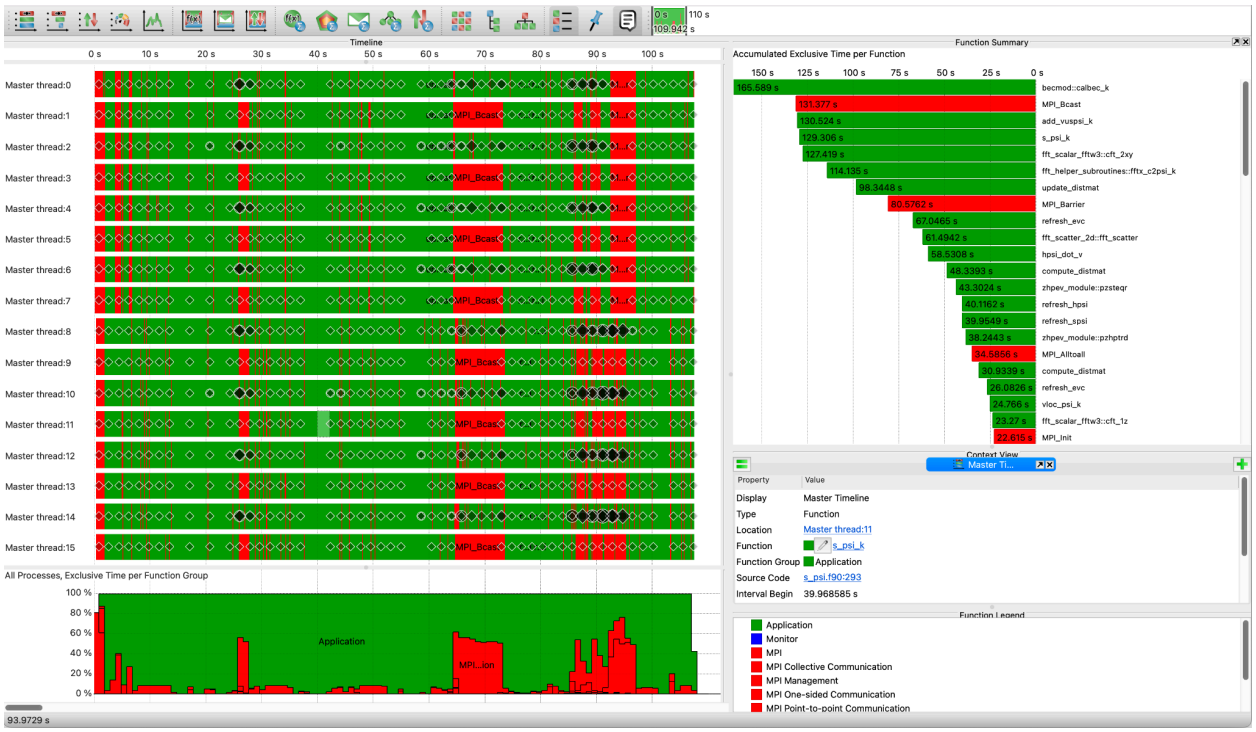

Figure 3: Visualizing trace data with Vampir.

Finally, trace data written to OTF2 files can be visualized with Vampir (Figure 3).[21,22] While the summary profiles provide only an aggregated view of the processes, tracing data provides time-stamped events, which is crucial for understanding the dynamic behaviour of the application, especially how much time is spent in wait states during MPI communication. The main window of Vampir is generally composed of the timeline (showing the detailed timeline of events for each process), the accumulated exclusive time per function, the context view (with more details about specific events), and the function view (listing all loaded function groups with a color-coded legend). Also useful is the summary timeline, giving a more general overview of how much time the application spends in each function group during different parts of the timeline.

21 https://vampir.eu/

22 Williams W., Brunst H. Parallel Performance Engineering using Score-P and Vampir. ICPE '23 Companion: Companion of the 2023 ACM/SPEC International Conference on Performance Engineering. Pp. 121 - 125. https://doi.org/10.1145/3578245.3583715, https://research.spec.org/icpe_proceedings/2023/companion/p121.pd

Another interesting aspect of Score-P/Scalasca is its energy efficiency measurement capabilities, which we plan to explore in the future and which could also be useful for WP4 activities.

3.3 Extra-P

Extra-P[23] is an open-source tool available under the 3-clause BSD license for an automatic performance-modelling to support the user in the identification of program parts that scale worse than expected. It uses measurements of various performance metrics at different execution configurations to create performance models. In order to search for scalability issues in full-blown applications it is sufficient to run five to six small-scale performance experiments per modelled parameter, launch Extra-P, and compare the asymptotic or extrapolated performance of the worst instances to the user's expectations. Usually, Score-P is used for this because Extra-P has built-in support for the call-path profiles of Score-P and the CUBE4 format. Alternatively, measurements taken with other tools can be converted to an Extra-P compatible file format, such as JSON or plain-text.[24]

Extra-P generates potential scalability predictions along with descriptions of models for any available performance metrics, such as computational operations (floating-point functions) or messages sent by MPI calls. This allows one to estimate requirements to finish specific tasks, answering questions such as the number of required MPI processes to finish a specific problem of a particular size in a fixed time frame and whether offloading a specific computation to the GPU can be efficient (considering the additionally needed data transfer) or will harm the overall performance.

We plan to further test Score-P and Extra-P capabilities in the area of GPU performance and gathering measurements on GPUs, since modelling and planning GPU acceleration in our applications remains a hard problem. Increased availability of such tools and inclusion in our benchmarking runs could provide further insights in effects of specific design choices of modular supercomputing architecture solutions on efficiency and scalability of mature codes and specific algorithms.

---

[23] A. Calotoiu et al. "Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes". In: Proc. of the ACM/IEEE Conference on Supercomputing (SC13), Denver, CO, USA. ACM, Nov. 2013, pp. 1–12. ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503277.

[24] More information about the file formats and the measurement requirements is available in the Extra-P repository: https://github.com/extra-p/extrap

3.4 ParaStation MPI

ParaStation MPI[25] implements DEEP-project extensions[26] supporting a wide range of applications in a fully MPI-3 compliant implementation (based on the MPICH-4.1.1 implementation). The extensions are designed to be as close as possible to the current standard, while still supporting the peculiarities of the DEEP prototypes. This way, applications tuned to modular system architecture environments can remain portable and essentially compliant with the MPI standard while still exploiting the underlying hardware capabilities.

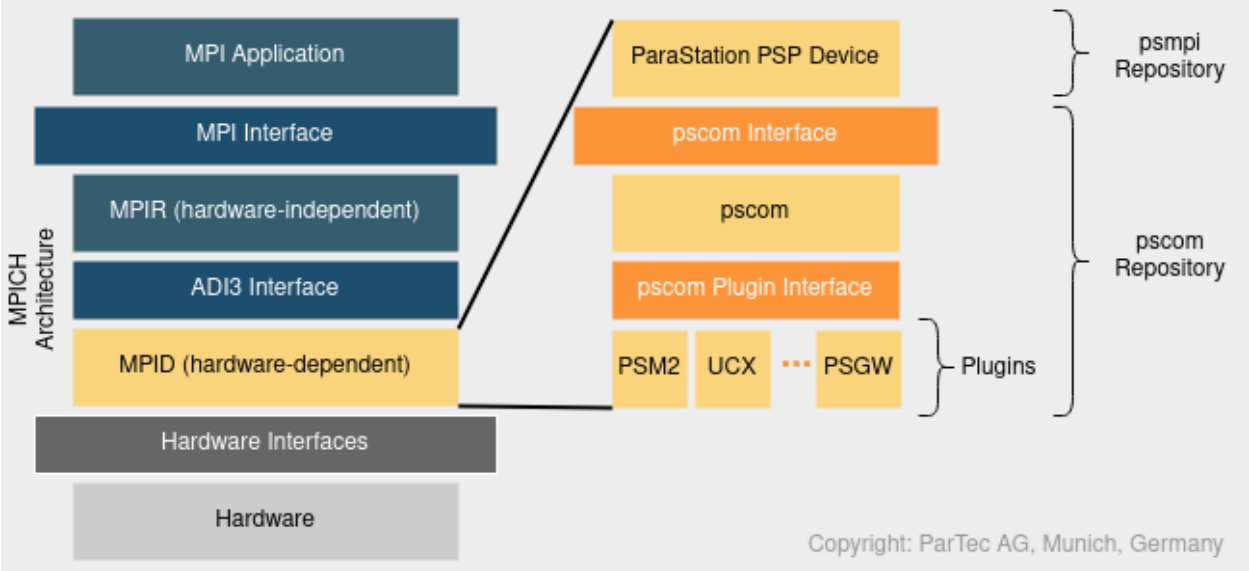

Figure 4. ParaStation MPI relies on the low-level communication layer pscom for the implementation of the PSP Device on the ADI-Layer of the MPICH architecture. (See e.g. https://github.com/ParaStation/psmpi)

ParaStation MPI can provide its modular system architecture awareness for better insight in the information collected. It supports a statistical analysis option, but in interaction with Score-P measurement system and/or the resource manager, ParaStation MPI can provide detailed information on I/O activities of an application.

But ParaStation MPI, In addition to the pure communication tasks that build the primary requirements of parallel application running on a cluster, provides further assistance of the runtime environment by providing a framework for cluster-wide job and resource management. The backbone of the ParaStation MPI management facility is built by the ParaStation MPI

[25] https://github.com/ParaStation/psmpi
[26] ParaStation MPI on DEPP Project site: https://deep-projects.eu/modular-supercomputing/software/programming-environments/parastation-mpi

daemons `psid`, running on every node of the cluster, communicating underneath each other constantly. Thus the management system is not located on a single node but distributed throughout the whole cluster. This design prevents the creation of single points of failure.[27] In addition, the system provides flexible tuning parameters for MPI and underlying interconnects as well as InfiniBand specific tuning variables with MPI tuning for connected service types - addressing scalability problems with InfiniBand. The system supports concurrent usage of different transports, with support for InfiniBand, Omni-Path, BXI, etc.

3.5 MPIGDB

MPIGDB[28,29] is a flexible debugging infrastructure for MPI programs. It is a serial debugger that enables attaching gdb to a PID of the application, it can be used to investigate deadlocked code, segfaults, and other errors for C/C++ and Fortran code. It can be used in combination with Valgrind.

3.6 Darshan

Darshan[30] is a scalable HPC I/O characterization and profiling tool designed to capture an accurate picture of application I/O behavior, including properties such as patterns of access within files, with minimum overhead. It specializes in instrumenting MPI application I/O and supports the most widely used HPC architectures. Additional modules support specialized filesystem instrumentation, i.e. for Lustre.

3.7 Valgrind

Valgrind is a well known debugging tool for analysis and detection of memory leaks and memory or pointer related errors in compiled applications with good support for parallel applications. It is well integrated in compiler suites and can be used together with gdb, by adding `--vgdb=yes` to the execution and then attaching gdb to the valgrind process. We recommend using Valgrind as a part of regular compilation or integration workflows.

We have performed some tests using interactive allocation:

---

27 Cluster Management Facility in ParaStation MPI Documentation: https://docs.par-tec.com/html/psmpi-userguide/ch02s02.html

28 https://github.com/robertu94/mpigdb

29 Underwood R., Nicolae B.: MPIGDB: A Flexible Debugging Infrastructure for MPI Programs. FlexScience '23, June 20, 2023. https://doi.org/10.1145/3589013.3596675 URL: https://hal.science/hal-04343674/document

30 https://wordpress.cels.anl.gov/darshan/

```
salloc --nodes=2 --ntasks-per-node=128 --cpus-per-task=1 \
       --time=00:20:00 –partition=cpu --hint=nomultithread
```

And then ran the pw.x executable from the Quantum ESPRESSO suite on a single node, checking for possible memory leaks:

```
mpirun -np 128 valgrind --leak-check=full \
       --show-leak-kinds=all pw.x…
```

Note that we have noticed possible false positives. The results of additional analysis require further studies since highly optimized advanced algorithms in hybrid OpenMP/MPI contexts with optional GPU support are challenging to fully analyse.

3.8 Likwid [31]

Likwid-tools[32,33] is a powerful command line performance tool suite for the GNU/Linux operating system. It supports performance testing and analysis tools for modern CPUs (Intel, AMD, ARMv8 and POWER9 processor) with additional support for some GPU architectures (Nvidia and AMD). It provides a wide array of tools for introspection, analysis, NUMA and core pinning, power measurement, counter and cpu feature flag access, frequency, power capping and cache finetuning.

Likwid is installed on the EuroHPC Vega machine. We have used likwid-pin to pin threads to specific NUMA nodes, which improved the performance of the MaX benchmarks. By using the likwid-pin tool, we optimised thread placement and memory access, which had a significant impact on performance – because of efficient cache usage, balanced tasks across all threads and because of minimised communication overhead (lower latency). On EuroHPC Vega, when applying the CPU affinity the performance of Quantum ESPRESSO improves about 20-fold. For example, the performance for the pre-processing step is as follows:

CPU pinning applied	CPU pinning not applied
13.9 seconds	273.6 seconds

[31] Likwid wiki: https://github.com/RRZE-HPC/likwid/wiki
[32] https://hpc.fau.de/research/tools/likwid/
[33] https://github.com/RRZE-HPC/likwid

These results are unexpected and suggest some other underlying issues on the cluster. However, repeating the benchmarks gives similar results. These figures are also likely to be influenced by other jobs running on the same worker nodes, as well as suboptimal task distribution. The default task distribution configuration should also be taken into account, as it does not always match the optimal settings for the workload. Therefore, such impressive improvements should not be expected in all environments – a 10-15% performance improvement is more likely. An important lesson learned, however, is that CPU pinning can not only improve performance, but can insure more predictable and sustainable performance in multi-tenant environments and under heavy load.

3.9 NVidia GPU Accelerator Developer Profiling Tools

NVidia NSight developer tools[34] and (to a lesser extent the older) NVidia Visual Profiler[35] have become the de-facto profiling tools for performance analysis of CUDA payloads. The NSight Systems[36] system profiler and the NSight Compute[37] as the compute kernel profiling tool are the cornerstones of analysis, and offered by many vendors as well as available for free from NVidia. In particular, profiling and analysis have been performed for benchmarking and development efforts at HPC Leonardo by Cineca.

We have found the use of these tools invaluable, but they remain limited to the NVidia platform, so they have to be complemented by other tools to support the specifics of other platforms.

3.10 AMD-upro

AMD-uprof[38] is a software profiling analysis tool, it provides performance metrics for AMD "Zen"-based processors, which are used on EuroHPC Karolina and Vega. AMD uProf includes multiple functionalities, such as performance analysis (CPU Profile), to identify runtime performance bottlenecks of the application, system analysis (eg. IPC and memory bandwidth) and power profiling to monitor thermal and power characteristics of the system.

Module AMD-uProf/5.0.1479 is installed on Vega and has been made available for testing and analysis.

[34] https://developer.nvidia.com/nsight-systems
[35] https://developer.nvidia.com/nvidia-visual-profiler
[36] https://docs.nvidia.com/nsight-systems
[37] https://docs.nvidia.com/nsight-compute
[38] https://www.amd.com/en/developer/uprof.html#documentation

4 Scheduling and Resource Management

4.1 Flux

Flux[39] is a next-generation resource and job management framework that expands the scheduler view beyond the single dimension of "nodes." Centralized resource managers such as Slurm or PBS Pro follow the traditional batch job paradigm: a few large, long-running, homogeneous jobs rather than ensembles composed of many, and often small, short-running heterogeneous tasks, i.e. "ensemble-based" workflows. Some of these present additional challenges: (1) massive throughput requirements, (2) sophisticated co-scheduling due to complex coupling of jobs, (3) complex job coordination and (4) portability, where user-supplied workflow managers fail to adapt to different HPC environments. In order to be able to address such challenges, Flux offers a composeable framework that enables new resource types, schedulers, and framework services to be deployed as data centres continue to evolve. The framework is designed with a *fully hierarchical* approach, giving Flux the ability to be seamlessly nested within batch allocations created by itself or other resource managers, simultaneously allowing user-level customization of scheduling policies and parameters.[40]

There are two primary components of Flux: workflows, which describe what actions or operations should run, when they should run, and how they relate to one another, and the engine, which drives the execution of workflows and determines when and how the workflows themselves are executed.

A workflow in Flux is composed of several pieces that describe what the workflow should do (and when). Actions allow one to perform an action (like running a process or Java code), triggers allow one to wait for a time, date, or event before proceeding, and flows describe how triggers and actions relate to one another (in other words, what order the triggers and actions should follow). Scheduling, therefore, is the use of triggers to specify when the other triggers and actions in your workflow should run.

[39] Flux: Building a Framework for Resource Management https://computing.llnl.gov/projects/flux-building-framework-resource-management

[40] D. H. Ahn, N. Bass, A. Chu, J. Garlick, M. Grondona, S. Herbein, H. I. Ingolfsson, J. Koning, T. Patki, et al., Flux: Overcoming scheduling challenges for exascale workflows, Future Generation Computer Systems 110, 202 (2020) https://flux-framework.org/publications/Flux-FGCS-2020.pdf

Fluxion,[41] implemented as the flux-shed module in the Flux framework, is an implementation of a composeable traditional scheduler in the flux framework. It allows for a user interface comparable to Slurm command line usage, but also allows for the management of payloads using workload configuration files or a python API. Fluxion exposes enough of the flexibility of the system to allow complex interdependent workflows to be executed and tested.

Flux is a modular infrastructure and seeks interoperability with SLURM. The example of deployment on the El Capitan system shows how usage of `flux_wrappers` and `mpibind` allows using Slurm command line constructs with flux.[42]

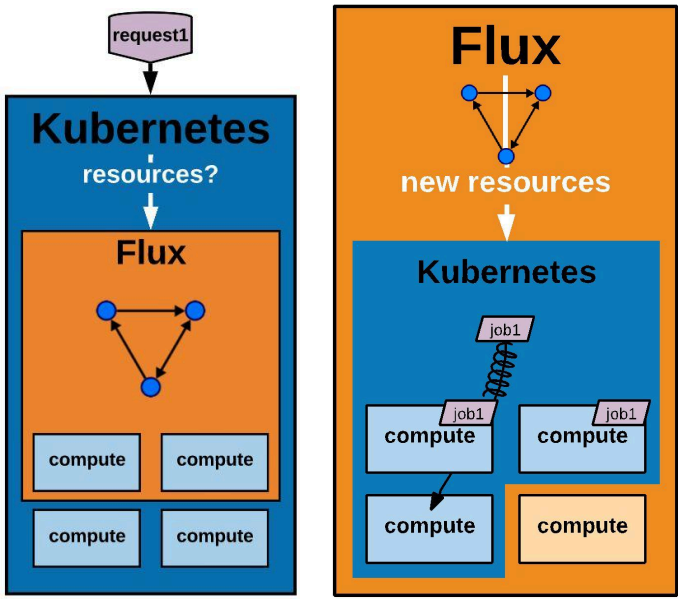

Figure 5. Flux and Kubernetes are fully interoperable: Flux can manage Kubernetes as the payload, can be the payload in Kubernetes clusters and can play the role of pod manager inside of a Kubernetes deployment, as desired.

[41] Tapasya Patki, Dong Ahn, Daniel Milroy, Jae-Seung Yeom, Jim Garlick, Mark Grondona, Stephen Herbein, and Thomas Scogland. 2023. Fluxion: A Scalable Graph-Based Resource Model for HPC Scheduling Challenges. In Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12-17, 2023, Denver, CO, USA. ACM, New York, NY, USA. https://doi.org/10.1145/3624062.3624286, https://dl.acm.org/doi/fullHtml/10.1145/3624062.3624286

[42] Using El Capitan Systems: Running Jobs with Flux and mpibind: https://hpc.llnl.gov/documentation/user-guides/using-el-capitan-systems/running-jobs-flux-and-mpi

Flux and Kubernetes. Flux has an extensive set of support features for deployment with Kubernetes in different roles, including as the pod manager/orchestrator (cf. Figure 5). We have not been able to perform in-depth testing and performance analysis at this time but recommend a study of viability of Flux and Kubernetes as a flexible and scalable highly performant and efficient convergent architecture.

The ability to build Flux with standard packages for dependencies or, alternatively, to run it from a container, makes it very easy to install. Furthermore, the ability to spawn Flux sessions under Slurm makes it very easy to experiment on a fully installed cluster without disturbing existing payload.
We have been able to perform a number of tests and experiments with Flux and flux-sched (Fluxion) on our local clusters and the EuroHPC Vega facility. Experiments with the hierarchical design of Flux have shown that the parallelism it provides and its facility of granting resources to children is indeed very efficient and helps managing job throughput challenges effectively. Experiments with non-uniform payloads have been non-conclusive, but optimistic. We do not offer efficiency measurements at this time since the tests were performed with Flux running under Slurm, so any speed limitations could be attributed to Slurm or their interplay.

4.2 Exascale Meta Scheduler: HyperQueue

HyperQueue[43] (HQ) is a specialised high-performance computing (HPC) job meta-scheduler developed to improve the efficiency, execution, and scalability of complex computational tasks. HQ enables load balancing of tasks across multiple available HPC resources, where it integrates local batch systems as middleware. It currently supports the prevalent workload management systems SLURM and PBS. This set-up allows for deployment along existing solutions on production infrastructure.
One of the most interesting features of HyperQueue is its advanced resource management capability.[44] HQ offers dynamic load balancing, which streamlines the distribution of tasks. Users are not required to pre-configure resources (e.g., allocating 16 CPUs) for specific tasks or manually bundle tasks into larger computational units. Instead, users simply initiate a worker node furnished with requisite resources, submit tasks, and HQ dynamically assigns those

[43] https://github.com/It4innovations/hyperqueue

[44] Böhm, S, et. al. (2021): HyperQueue: Overcoming Limitations of HPC Job Managers, SC 2021: https://sc21.supercomputing.org/proceedings/tech_poster/tech_poster_pages/rpost104.html

resources as needed. Additionally, workers need not be launched by hand; HQ possesses the capability to autonomously initiate SLURM or PBS jobs on demand.

The core feature HyperQueue brings on the table is the ability of complex resource management and load balancing of tasks across all available HPC resources. HQ provides automatic submission of Slurm/PBS jobs on behalf of the user, supports complex and arbitrary task resource requirements (number of cores, GPUs, amount of memory, other hardware requirements, such as FPGA accelerators etc.), allows for selection of non-fungible resources (so that tasks can be assigned to specific resources, e.g. a GPU with ID 1), supports resource variants (tasks can require alternative solutions, e.g. "`1 GPU and 4 CPU cores`" or "`16 CPU cores`") and fractional resources (tasks can require e.g. 0.5 of a GPU) as well as related resources (tasks can require e.g. "`4 CPU cores in the same NUMA node`").

In addition, HQ scales to hundreds of nodes/workers and millions of tasks with a very small overhead per task (below 0.1 ms) and allows for direct streaming of stdout/stderr avoiding stress on distributed filesystems by creation of small output files. HQ has a set of simple and flexible user interfaces: task graphs can be defined using command line options, TOML workflow files or using a Python API. Cluster utilization can be monitored with a real-time dashboard.

HyperQueue is an open-source Rust project released under the MIT licence. It is very easy to deploy – it is packaged as a single, statically linked binary without any runtime dependencies (apart from `libc`). No administrator access to cluster configurations is needed to deploy since HQ uses a natively installed scheduler API to manage the cluster.

Detailed results and analysis of our work with the exascale meta-scheduler are described in detail in a separate deliverable: **D2.3 Report on the exascale meta scheduler**.

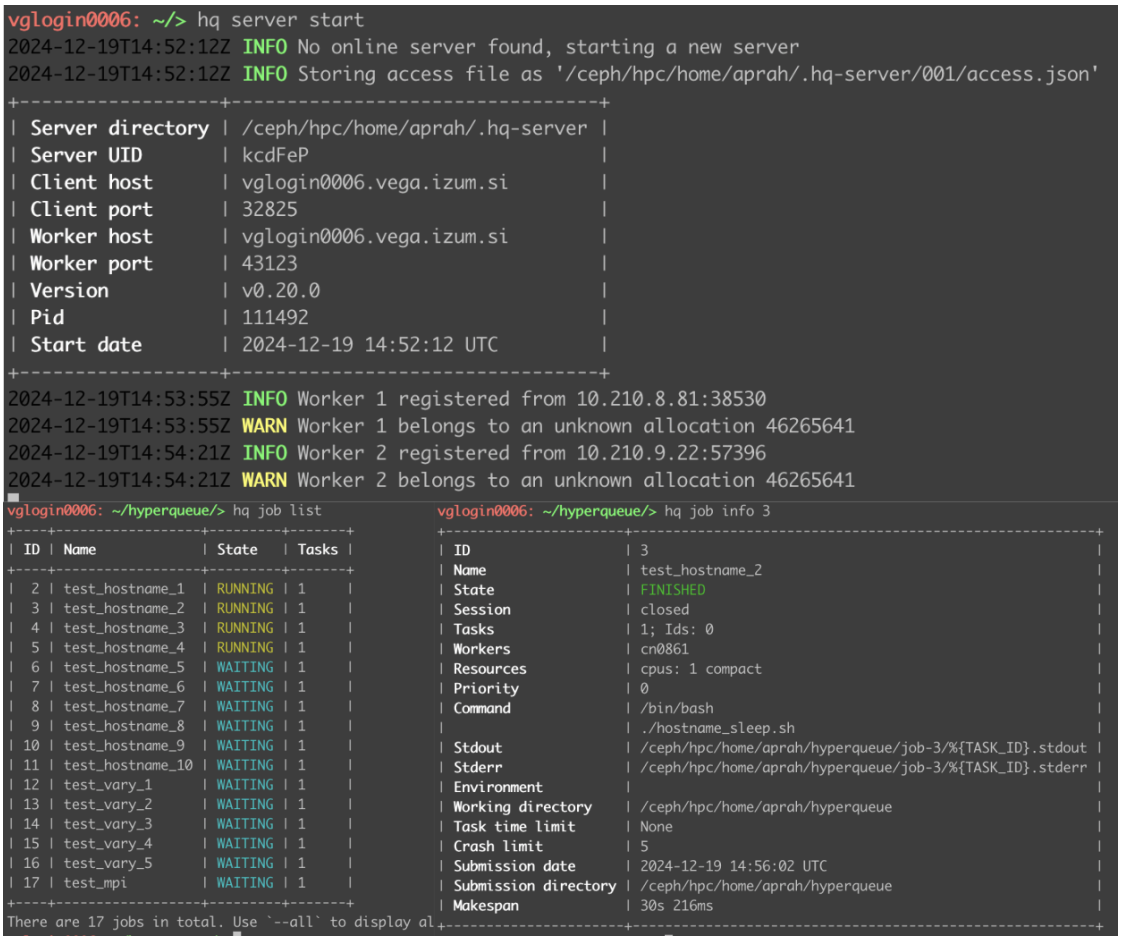

Figure 6. Running a HyperQueue server to submit some test jobs on HPC Vega.

4.3 Slurm and Kubernetes

The motivation behind combining Slurm and Kubernetes in the context of MaX codes comes from two different contexts: in cloud-based commercial deployments, Kubernetes has become a prevalent player, and being able to use it to schedule the computational load is an important advantage for a code-base that expects to scale to very large systems. On the other hand, development towards efficient exascale workflows require a number of solutions that are difficult to represent as computational load since they mostly have the nature of a persistent service: work-flow management, intermediate data storage, databases, CI/CD systems, monitoring and observability systems. These solutions are often developed and prepared to be deployed on container orchestrators, such as Kubernetes.

Both Slurm and Kubernetes can be used for managing processing load or resource orchestration. In particular, Slurm is optimized for high-performance computing (HPC) in general, assuming that infinite batches of finite, but potentially large-scale concurrent loads (MPI/OpenMP) are available in a context of a fixed system configuration, while Kubernetes has not been designed for running time-finite workloads, but for running conditionally infinite microservice applications in cloud-native systems where resources are also assumed to be "infinite" for autoscaling.[45]

Slurm and Kubernetes solve similar problems and offer comparable services but there are a number of differences and trade-offs. In addition, both systems rely on modern Linux kernel features but use them very differently, and expect very different management contexts on the nodes.

Kubernetes has become the de-facto standard resource orchestrator on the majority of public clouds providing computational resources, while Slurm became the most prolific HPC orchestrator (advanced batch scheduler) for new installations. As such, SLURM has sophisticated features targeting HPC, deep control of hardware resources using modern kernel facilities and is highly extensible, including container support (i.e. Singularity/Apptainer and Pyxis from Nvidia, see next section) but lacks strong auto-scaling features and requires stable filesystems and well defined hardware nodes making it less suitable for deploying in cloud environments where elastic scalability is required.

Kubernetes, on the other hand, has a control plane that is considerably more complex than Slurm's, requires any payload to be containerized and requires a lot more configuration to support the payload. All parts of the infrastructure need to be represented as Kubernetes resources, so the deployment of a complex workload can be daunting. The advantage of the system is its universality: workloads very different from typical web services or HPC jobs can be represented and managed. With proper configuration, *auto-scaling* and *self-healing* as well as *high availability* are among the features offered out-of-the box. However, Kubernetes can't boast of advanced scheduling, can't offer granular management of hardware and lacks support for HPC-specific features (Idem.) However, many of these draw-backs can be mitigated through third-party extensions, specifically additional Kubernetes operators installed in the cluster. The list is usually quite long and these days, with GPU and ML support as main targets, usually

[45] Mikhail Mokrushin; *Slurm vs Kubernetes: Which to choose for model training.* Nebius blogs. https://nebius.com/blog/posts/model-pre-training/slurm-vs-k8s

consists of the MPI Operator,[46] NVidia GPU Operator,[47] Nvidia Network Operator,[48] some solutions for shared filesystems and databases, etc. This significantly complicates the bootstrapping and managing of the system.

There has been extensive research and reporting on experiments on combining these approaches to infrastructure management. In general, we can consider the main options as **shared deployments**, **Slurm-first or Slurm-over and Kubernetes-first or Slurm-under.**[49] In the **Slurm-first** or **Slurm-over** mode, the cluster resources are managed by Slurm, Kubernetes clusters are created ephemerally within Slurm batch jobs and therefore Kubernetes control plane is either unavailable until job launches or needs to be hosted outside of the traditional cluster. In the **Kubernetes-first** or **Slurm-under** mode, the cluster resources are managed by Kubernetes, with Slurm as the payload. An example of these solutions is Nebius Soperator: Kubernetes Operator for Slurm.[50] The advantage of this approach is the use of autoscaling and self-healing of Kubernetes in Slurm, and the ability to use Slurm in environments that are typically managed as Kubernetes clusters.

Soperator (free, under Apache licence 2.0) supports automatic shared root for the Slurm payload, GPU health checks, high scalability and availability as well as user isolation. Slurm contributes accounting. NVidia GPU Operator and NVidia Network Operator are required, non-GPU nodes are not supported and there is no support for running outside Nebius AI clusters – so we have explored this truly as an example of a kubernetes-first setup. There are many similar solutions. In general they can not provide the fine granularity of hardware resource use provided by native Slurm and require non-native scheduling of resources. Shared deployments are meant to overcome such limitations **Shared deployments** can be implemented as **distant** or **adjacent**.

With the **distant shared** deployment, we mean that Slurm and Kubernetes schedulers are not aware of each other, the current resources and demand for the other environment. Kubernetes and the HPC workload manager coexist on the same cluster, but it is difficult to have the cluster be fully utilized. Either resources have to be throttled or another management tool has to shift nodes between the two schedulers. In the share deployment option, where HPC and container

[46] MPI Operator: https://github.com/kubeflow/mpi-operator

[47] NVidia GPU Operator: https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/index.html

[48] Nvidia Network Operator: https://github.com/Mellanox/network-operator

[49] Terminology from presentation Slurm and/or/vs Kubernetes by Tim Wickberg at SC2023, https://slurm.schedmd.com/SC23/Slurm-and-or-vs-Kubernetes.pdf

[50] Nebius Soperator: Kubernets Operator for Slurm: https://github.com/nebius/soperator

workloads are supported natively in the same shared environment, users should be able to choose the environment appropriate to their workload or workflow type. One approach to supporting mixed workloads is to allow Kubernetes and the HPC workload manager to co-exist on the same cluster, throttling resources to avoid conflicts, but preventing the cluster from being fully utilized. If full utilization is to be achieved, the recommended approach is to use a *peer scheduler* that coordinates with the Kubernetes scheduler. An example of a distant shared deployment (without a peer scheduler) is Dell's Omnia.

Omnia, an open source (Apache-2.0 license) supports both Slurm and Kubernetes nodes.[51] The tool uses Ansible playbook-based methods to deploy and manage a Slurm/Kubernetes cluster on servers running an RPM-based Linux OS with complete management and supervision support. While Omnia is an open source community project, it started in the Dell Technologies HPC & AI Innovation Lab and is commercially supported by Dell for its products. Omnia can compose the solution stack to support a variety of workload with an infrastructure-as-code approach, including the set-up of the NVidia GPU and AMD ROCm platforms, BeeGFS as a hardware-independent fast POSIX parallel file system storage and a number of pre-packaged solutions for classical HPC and modern AI payloads.

It has, however, a number of limitations. First among them is relatively sparse support for non-Dell configurations. In addition, it does not support container payloads on Slurm out of the box,[52] and only support one storage instance (Powervault) in the HPC cluster. There are limitations to the number of auto-configurable nodes to 1000 and the types of automatically configurable switches (only Dell and NVidia switch configurations are included).[53] Monitoring, user management, authorization and authentication are centralized, but no automatic resource allocation between Slurm and Kubernetes is provided: the administrator needs to reprovision the nodes if they are to be allocated for the other resource manager, which makes the solution less flexible than desired.

---

[51] Omnia, Dell's Ansible playbook-based deployment of Slurm/Kubernetes clusters on servers running an RPM-based Linux OS: https://github.com/dell/omnia

[52] Out of the box; this can be enabled using Singularity/Apptainer: https://infohub.delltechnologies.com/en-us/p/containerized-hpc-workloads-made-easy-with-omnia-and-singularity/

[53] Omnia open-source software solution overview: Run AI, HPC and data analytics on the same systems, with greater flexibility. Dell 2022. https://www.delltechnologies.com/asset/en-za/products/ready-solutions/technical-support/omnia-solution-overview.pdf

The **adjacent shared** deployment overlaps both control planes and requires direct support from the resource managers, usually in the form of a Slurm plugin. **Slurm's Kubernetes scheduler plugin** allows Slurm to prioritize and schedule both Slurm and Kubernetes workloads: Kubernetes jobs are managed by the kubelet with full access to Kubernetes capabilities and Slurm jobs run through Slurm with the usual management for high-throughput and large-scale MPI workloads with the traditional HPC command-line interface. However, Kubernetes scheduling is still at node-level granularity and some of the Kubernetes scheduling primitives are difficult to represent in Slurm. (Idem.)
ShedMD, the Slurm company, is currently (November 2024) releasing a more flexible adjacent share solution under the name **Slinky**, a set of powerful integration tools designed to bring Slurm capabilities into Kubernetes. **Slurm Operator** plug-in becomes the central element,[54] with the future Slurm Bridge promised to enable automatic scheduling of workloads across a Kubernetes cluster. This is said to allow for co-location of Slurm and Kubernetes workloads, bringing in the advantages of the Slurm scheduling and scale to both. We will test these solutions in the following year when they become available.

We have been able to explore a version of hybrid resource management deployment through our collaboration with the **InterTwin project**.[55] InterTwin aims to co-design and prototype an interdisciplinary Digital Twin Engine based on a co-designed Blueprint Architecture.[56] The software stack, currently comprising 38 components, is developed open on GitHub.[57] The first edition of the prototype InterTwin engine is available at the InterTwin web site.[58] Among the components, the Core DTE module *Infrastructure Manager*[59] is used to deploy an application or service in a cloud environment, either composed by VMs or by Docker containers, on either public IaaS Clouds (Amazon Web Services, Microsoft Azure, etc.), on-premises cloud management platforms (OpenNebula, OpenStack, etc.) and container orchestrators (Kubernetes) with both horizontal (adding/removing nodes) and vertical (growing/shrinking the capacity of nodes) elasticity management. The *interLink*[60] federated compute module developed

[54] Slinky at ShedMD: https://www.schedmd.com/introducing-slinky-slurm-kubernetes/ , https://www.schedmd.com/slinky/why-slinky/
[55] https://www.intertwin.eu/
[56] The current version of the architecture is available on Zenodo: https://zenodo.org/records/10650440
[57] https://github.com/interTwin-eu
[58] https://www.intertwin.eu/intertwin-digital-twin-engine/
[59] https://github.com/grycap/im
[60] https://github.com/interTwin-eu/interLink

in the context of interTwin project provides an open-source solution capable of extending the container orchestration de-facto standard (kubernetes) to support offloading to any type of resource provider (Cloud/HTC/HPC) in a transparent manner, where little to no knowledge is required for the end user. In our example, controlling components managed at Cineca have been set-up to deploy compute payloads with datasets to the EuroHPC Vega system to be automatically provisioned and executed under Kubernetes, running on the main computing cluster (normally accessed through Slurm).[61] The set-up has proven high efficiency, scalability and, above all, robustness of this hybrid solution, but also served as an example of a cloud-native federated access to a HPC resource.

4.4 Arax

Arax: A Runtime Framework for Decoupling Applications from Heterogeneous Accelerators,[62] implements dynamic mapping of application tasks to available resources, and provides management of all required task states, memory allocations, and task dependencies in order to be able to not only share accelerators across applications in a server, but also adjust the resources used by an application as the load fluctuates over time and an application migration mechanism to enable re-balancing of accelerator load.

Arax offers a simple API that targets application developers, but it also includes Autotalk, a stub generator that can automatically generate stub libraries for applications written for a specific accelerator type, such as NVIDIA GPUs. In this way, Arax applications are written once without considering physical details, including the number and type of accelerators, and are more flexible and portable, but Arax also offers the same flexibility and portability to existing applications when using Autotalk.

Arax developers find 12% overhead for typical applications, such as Caffe, TensorFlow, and Rodinia, but show that Arax accelerator sharing is very efficient, by offering up to 20% better execution times compared to NVIDIA MPS, which only supports NVIDIA GPUs.

Arax applications issue tasks to task queues (similar to CUDA/ROCm streams and OpenCL command queues), but these queues are not assigned directly instead, Arax assigns them dynamically at runtime.

---

[61] C.f. InterTwin D5.2 - First DTE Infrastructure Software release. https://zenodo.org/records/10224150 and D5.1 First Architecture Design and Implementation Plan: https://zenodo.org/record/8036983

[62] Manos Pavlidakis, Stelios Mavridis, Antony Chazapis, Giorgos Vasiliadis, and Angelos Bilas. 2022. Arax: a runtime framework for decoupling applications from heterogeneous accelerators. In Proceedings of the 13th Symposium on Cloud Computing (SoCC '22). Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3542929.3563467, https://arxiv.org/pdf/2305.01291

We have examined the evaluation of Arax for its default targets, the ML workloads. We paid specific attention to Autotalk which allows for automatic porting of applications with acceptable overhead while manual porting of applications shows marginally better results but requires adapting the source. We consider the approaches that enable application migration and better utilization crucial for future exploitation for accelerator-based exascale machines, but have not envisioned adaptation of MAX codes to the Arax framework. We will follow the work and exploit possibilities of Autotalk-based deployments with Hosting Entities. In addition, we consider testing Arax as a tool to exploit adapting applications to different accelerators (i.e. ROCm or Neon).

The goal of Arax is improved utilisation of accelerators in modern systems. Studies[63,64,65] show that only 20% of jobs manage to use more than 50% of available SM (streaming multiprocessor) of a single GPU, only 4% of jobs use more than 50% memory compared to utilization and only 15% of jobs use more than 50% of the total available memory size. By providing efficient and safe accelerator multi-tenancy together with elastic scaling, Arax should be able to improve utilization and efficiency of GPU accelerators. Additional work, such as the G-Safe[66] library for safe GPU sharing in multi-tenant environments. From the point of view of application and library developers, we have to make sure our work is compatible with these solutions in order to enable better utilization and therefore better scalability while using very large resources once this approach to improving resource utilization should become wide-spread.

4.5 New features in Slurm

With the explicit design goals of an efficient, modern resource manager, Slurm became the prevalent batch/resource manager on modern machines. The design has been very successful due to its reliable efficiency: the back-fill scheduler and advanced scheduling policies allow administrators to set-up efficient exploitation policies for modern machines and supports

[63] NSDI'22, MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters, Alibaba Production Cluster

[64] HPCA' 22, AI-Enabling Workloads on Large-Scale GPU-Accelerated System: Characterization, Opportunities, and Implications, MIT Supercloud

[65] Arxiv'17, Workload Analysis of BLUE WATERS, NCSA Petascale-level supercomputer

[66] M Pavlidakis, G Vasiliadis, S Mavridis, A Argyros, A Chazapis, A Bilas. G-Safe: Safe GPU Sharing in Multi-Tenant Environments. arXiv preprint arXiv:2401.09290, https://doi.org/10.48550/arXiv.2401.09290

modular architectures, dedicated accelerators and fine-grained allocation. Its support for kernel-based control using cgroups and namespaces is robust and efficient.[67]

These features and design decisions, however, make it challenging to support advanced workflows and container-based workflows. Advanced workflows can represent a significant load on the system, and due to increased load in very large systems, can become problematic since the system favours management requests over scheduling work in order to remain responsive. Container-based workflows are challenging since the use of cgroups and namespaces as well as the authentication and management models can clash.

Overall, new features in Slurm support more manageable and efficient resource management on very large systems. Slurm development has included design decisions, support for integration in hyperconverged deployments as well as innovations for scalable workflow management, all of which is instrumental in supporting our payloads on exascale machines.

4.5.1 Step Manager

In Slurm, a job represents an allocation of resources whereas a step is a set of tasks which use a subset of the job resources: steps typically run parallel programs (e.g. MPI) and can run serially or concurrently. In this sense, a step can be a *batch step* (an instance of a batch script), an *interactive step* or an *extern step* (prologs, set-up of X11 forwarding, containers, tracking services started outside of an allocation etc.) and a *job step* (job.01, job.02 etc.), executing and taking care of a workflow process. Management of steps therefore becomes instrumental for complex workflow support in Slurm. Step management has been a source of congestion for Slurm since it requires a job management lock - blocking a highly threaded, but not highly concurrent system.

This has been resolved by the introduction of an independent Step Manager: in modern versions of Slurm, `srun` sets up I/O and communicates with `slurmd`, while `slurmd` sets up credentials and forks an instance of `slurmstepd`, the Slurm Step Manager. The Step Manager is responsible for connecting I/O, launching and supervising tasks and notifying `slurd` and `srun` on termination. Step Manager is configurable not just globally, but also per job or using a lua script and can be controlled using `scontrol`. This makes modern Slurm deployment much more flexible for intensive workloads, including heterogenous job loads. `slurmctld` still tracks batch, extern and interactive steps as well as heterogeneous jobs, but one extern step is

[67] Jette, M.A., Wickberg, T.: Architecture of the Slurm Workload Manager. In: Klusáček, D., Corbalán, J., Rodrigo, G.P. (eds.) Job Scheduling Strategies for Parallel Processing, pp. 3–23. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-43943-8_1

now designated as the Step Manager and it becomes responsible for communication with `slurmd`, `slurmctld` and client commands. The Step Manager tracks job_steps and relieves the controller of the load related to stop management. In this way, the design is closer to the advantages provided by the hierarchical design of the Flux resource manager toolkit.
Currently, it is reported that Slurm step launch throughput is directly related with job launch throughput, peaking around 300 job (and step) starts per second. With the new set-up using the Step Manager, the goal is to sustain over 500 job starts per second, where each job is independently capable of launching over 1000 steps. This provides sufficient flexibility and scalability for Slurm to be able to support API-based task management, advanced task management stacks and to accommodate specific requirements of some modern very large system stacks (i.e. Cray MPI port requirements implemented for all steps).

We see this step manager improvement as critical to supporting more fine-grained task management using Slurm. Currently, complex tasks are difficult to manage directly using the HPC middleware: using them within the allocation makes it impossible to take advantage of the scheduling facility to improve resource utilisation and time-to-result for scientific loads, while running task management outside the HPC middleware is often not well supported at sites. We believe that the step manager subsystem should be configured globally. By using the workload managers mentioned above, performance issues can be confined to the node where the job is running. MPI jobs can also benefit from this configuration. We will test this feature on national infrastructures as it is relatively new and has currently not been implemented on any EuroHPC cluster.

4.5.2 Singularity and news with Container Support

SISSA provides optimised Quantum Espresso GPU containers that are available also on Nvidia Cloud (NGC). Performance results show that using the container does not differ from the bare metal performance. This demonstrates how well the modern container architecture with well developed support in Slurm and other orchestrators can be used to exploit the hardware capabilities.

4.5.3 Pyxis plugin and Cloud-Native Interfaces

Pyxis[68] is a plugin for the Slurm workload manager that allows unprivileged cluster users to run containerized tasks through the `srun` command, including fast Docker image download,

[68] NVIDIA. Pyxis github repository. https://github.com/NVIDIA/pyxis (accessed December 17, 2024)

support for layers caching, layers sharing across users and package installation inside containers. Support for MPI jobs is provided via Slurm interfaces. Pyxis uses **enroot**[69] to manage containers and provides a way to use Slurm with cloud-native payloads to deliver the user-environment software stack exclusively, quickly, and transparently via containers to users while seamlessly integrated with the capabilities of an HPC system. Advanced cloud-native deployments, i.e. with Slurm managed under Nomad with Podman and the raw_exec executor, deploying GitLab runners and HPC tasks in the same environment.[70] NVidia recommended GPU containerized solutions include combining NVIDIA Container Toolkit[71,72] with its container running library and utilities to automatically configure containers to leverage NVIDIA GPUs, enroot and the pyxis plugin to provide near-seamless integration of containerized GPU-aware payloads with HPC environments.

Clearly, the motivation behind this is the LLM training load, where Slurm's ability to provide good hardware utilisation and resource management is an important motivator. And while the challenge of building containers optimized for the hardware infrastructure remains and negatively impacts the goal of portability and hardware independence of container runtimes, the integration and ease of use make these solutions flexible and efficient. The use of enroot avoids many pitfalls of trying to manage combined Docker na Slurm deployments and helps with the integration of the container infrastructure with Slurm and HPC. Due to the push from the vendors and the AI community, we can assume that further development in this direction will provide better solutions of containerised hardware-aware deployment in HPC infrastructures.

4.6 mpibind and Slurm

Mpibind is a memory-driven algorithm to map parallel hybrid applications to the underlying architecture transparently from the point of view of the application. It is implemented in a library that employs a simple interface for computational scientists. The library provides a full mapping of MPI tasks, threads, and GPU kernels to hardware processing units and memory domains. It is designed to deal with increasingly complex modern systems that include many-core technologies, machines with hybrid cores combining performance- or

[69] NVIDIA. Enroot github repository. https://github.com/NVIDIA/enroot (accessed December 17, 2024)

[70] Cruz, F.A., Dabin, A.J. and Ballesteros, M.S., 2023. Deploying Cloud-Native HPC Clusters on HPE Cray EX. In proceedings of the Cray User Group (CUG) conference. URL: https://cug.org/proceedings/cug2023_proceedings/includes/files/pap131s2-file1.pdf

[71] https://github.com/NVIDIA/nvidia-container-toolkit

[72] https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/index.html

throughput-optimized cores and latency-optimized cores, and supports not just NUMA, but multiple levels of memory. Providing an automated, algorithmic approach to mapping of application requirements to specific underlying hardware resources has become a crucial problem due to raising complexity and rapid development of complex and modular solutions. Furthermore, developers and users can avoid dealing with intricacies and details of hardware topology and concentrate on the algorithms and tasks.
With the SPANK plugin, mpibind is very easy to incorporate in the usage of a Slurm system. It allows us to combine ease of use with optimized high performance across different computer architectures and within a well understood standard interface.

5 Workflow Managers and Deployment Interfaces

5.1 AiiDA

AiiDA is an efficient platform for workflow development based on the AiiDA engine and code-specific plugins and was developed and used extensively during MaX-1 and MAX-2, mostly for high-throughput computing and the associated high-performance data analytics. AiiDA workflows exploit the core features provided by the platform but are enhanced with plugins for MaX codes that encapsulate the functionality of the codes and expose their I/O as AiiDA-compatible data structures. AiiDA stores all relevant nodes in a dedicated database, enabling provenance tracking for calculations by maintaining full provenance logs of the calculations performed.
AiiDA has been primarily used as a high-throughput computing solution for relatively small calculations with relatively restricted mutual data dependencies, but can be used as an orchestration tool for exascale workflows with some relatively small extensions: (1) an interface to code performance models to automatically calculate and inject resource options at each relevant step (replacing manual specifications); (2) higher task-launching launching semantics, using existing HyperQueue interface, to provide the abstraction of "workers"; (3) adapt provenance tracking to only log incoming and outcoming data to allow for extreme scalability; (4) refactor MAX code plugins for task and data offloading and (5) support remote data abstractions for data migration and integrate with HyperQueue feature support and possible future EuroHPC federation data management features.
We planned to include features for conditional and coarse-grained processing of peta- to exascale tasks and direct interfacing with the logic of the AiiDA orchestrator as an additional measure to support exascale workflows that could otherwise saturate unevenly the schedulers

or overload the system resources as well as implement new AiiDA interfaces for data handling, to enable support for fast data stores beyond the traditional filesystem paradigm (c.f. section **Fast Distributed Storage**). This will be needed to support exascale workloads once they bring much larger datasets, with the need to decide at runtime what to persist and what to process in order to avoid the I/O and storage layers becoming the bottleneck of the code execution. This will benefit by the new data interfaces implemented in AiiDA. (Please refer to Task 2.3: Orchestrators and AiiDA management for exascale workflows for detailed information.)
As reported in the MAX D2.1 Deliverable,[73] the platform has proven its scalability by a run on the entire LUMI-C partition of 196608 AMD Epyc cores, coordinating 55704 Quantum ESPRESSO calculations for 15324 inorganic compounds, with essentially 100% utilisation over 12 hours. The main observed drawback was execution of AiiDA workflows due to enhanced security measures on sites, in particular different implementations of two-factor-authentication schemes that complicate technical approaches to remote execution. Unfortunately, the problem is essentially a legal security and insurance issue, therefore seeking of simple technical solutions will not suffice to provide good workflow support on modern machines. The issue has been raised with hosting entities and EuroHPC and is being discussed. The same issue has also occurred with other initiatives, such as data management and CI/CD platform support. Not running a workload manager as a service has another drawback, since the lifetime of the manager depends on the lifetime of the user credentials and job allocation time, which limits the usefulness of the software considerably, to the extent of rendering it futile.

5.2 RemoteManager

RemoteManager is a modular serialisation and management python package for handling the running of functions on remote machines. While it is a flexible general utility that enables deployment and remote execution of workflow components, it can be thought of as a lighter alternative to the corresponding AiiDA module. It features robust "front-end" capabilities linked to notebooks and permits users to use notebooks to interact with remote data and compute payloads in HPC centres. The package is developed within the BigDFT effort as a general purpose implementation of the BigDFT RemoteRunner concept with considerable improvements and expansions on that conceptBased off of the BigDFT RemoteRunner concept, remotemanager represents an improvement and expansion on that concept.

[73] MAX D2.1 Deliverable: Exascale Workflow Design (Sunday, 31 December, 2023, at the MAX Project Repository)

Primary usage of RemoteManager is centered on the **Dataset**, connected to a remote machine using an **Url** object. The **Dataset** can be used as a "container" for a calculation and its data, to which individual "runs" are attached. The data and the runs are located and executed on the remote machine described by the provided **Url** object, which means that the interactive component and the calculating kernels are seamlessly connected, but running in disjoined environments.
Current implementations overcome significant weaknesses of many similar solutions, including our initial PyBigDFT implementation, where workflow writing and management was difficult since the workflow had to be interrupted in order to offload computationally heavy calculations to remote machines. For workflows written in a Jupyter notebook, one would need to convert that notebook to a script, submit it through the job system, wait for the calculations to complete, copy data to the machine with the Jupyter server, and then restart the notebook (exploiting lazy calculations). However, computationally demanding workflows can include many different intensive operations, pre- and post- processing steps etc. The `remotemanger` library solves both of these problems: it can serialise arbitrary Python routines and arguments and send them to the remote machine in order to be executed through the job system, collecting the data and synchronising it to the local machine. It is fully extensible via `@RemoteFunction` decorator and due to optimization of synchronisation steps also usable for data-intensive use. Its connection methods support ssh-based connections and we have shown that it can overcome challenges of two-factor authentication.
RemoteManager has comprehensive support for notebooks which makes it a suitable tool for exploratory calculations, testing and teaching/collaborative work using literate programming (with examples and documentation included). RemoteManager can be used as a general RPC library for HPC systems to enable exploration in the context of large data and computational requirements.

Within the MAX project, the library is currently used primarily by the BigDFT team, but its versatility allows it to support tasks such as benchmarking and validation. We plan to perform further testing and also to promote a more general use of the library within MAX and offer its solutions to wider communities.

6 Data Handling

6.1 DLB library

BSC Dynamic Load Balancing library[74] is a dynamic user-transparent library that improves the load balancing of hybrid applications and manages the number of threads, and is compatible with MPI, OpenMP, and OmpSs[75] with additional modules LeWI (Lend While Idle), DROM (Dynamic Resource Ownership Management), and TALP (Tracking Application Live Performance).

DLB offers advanced features when integrated with the OmpSs-2 runtime, including support for task nesting and fine-grained dependencies across nesting levels, which enables effective parallelisation of applications using a top-down methodology. (We were not able to test these features at this time since we experimented on clusters where no OmpSs-2 is provided, but will schedule testing with colleagues from BSC.)

In addition, the DLB library allows the use of OMPT interface to the OpenMP runtime to run as an OpenMP performance tool that can monitor and gather performance data in real-time, as a very useful observability and monitoring tool. In addition, by using the DLB interface, an application can choose to release some CPUs used for OpenMP threads to the resource manager, if the measured parallel efficiency is under certain value, making it possible to deploy more efficient and flexible workloads.

7 Fast Distributed Storage, Object Stores and Fast Databases

Modern large scale payloads and workflows in materials design require intensive data exchange that can be managed via the filesystem, which is still the easiest to understand and use, but special demands require occasional reliance on specialised tools. We expect to eventually prepare interfaces which are semantically similar but more efficient under specific constraints.

[74] DLB: Dynamic Load Balance library. URL: https://pm.bsc.es/dlb

[75] M. Garcia, J. Corbalan, and J. Labarta. “LeWI: A Runtime Balancing Algorithm for Nested Parallelism”. In: Parallel Processing, 2009. ICPP ’09. International Conference on. Sept. 2009, pp. 526–533. DOI: 10.1109/ICPP.2009.56.

(1) Burst buffers[76,77] positioned as an intermediary storage layer between the front-end computing processes and the back-end storage systems bridge the performance disparity that can appear between the processing speed of compute node. (2) BeeGFS On Demand[78] is an effective solution providing a fast ad-hoc filesystem active only for the duration of a simulation as a fast-store for initial staging of data or a scratch area. (3) Apache Cassandra provides a distributed, wide-column NoSQL database store and management system designed to handle large amounts of data across a network of commodity servers with high availability and no single point of failure, and is a suitable solution when data has to be processed out of order from different parts of the workflow

In addition, development of an AiiDA data transfer layer for direct data exchange is under way in order to provide an option for bypassing filesystem bottlenecks. This abstract layer will interface with the appropriate store technologies considered in this task and will be used to support exascale workloads once they bring much larger datasets, with the need to decide at runtime and based on workflow specifics what to persist and what to process in order to avoid the I/O and storage layers becoming the bottleneck of the code execution. This will benefit by the new data interfaces implemented in AiiDA.

7.1 Weka

Weka[79] differs from legacy storage systems, software-defined storage solutions and traditional parallel file systems in that it overcomes traditional storage scaling and file sharing limitations while also allowing parallel file access via POSIX, NFS, SMB, S3, and GPUDirect Storage. A commercial product and part of the WEKA Data Platform, WEKA's parallel file system is designed to provide a cloud-like experience so that one can run their applications seamlessly on premises or in the cloud. Its distributed, parallel file system avoids the traditional block-volume layer managing underlying storage resources. This integrated architecture does not suffer the limitations of other shared storage solutions and delivers both scalability and performance effectively. It is implemented in a kernel driver and a number of services that can run as user

[76] J. Bang, A. Sim, G. K. Lockwood, H. Eom and H. Sung, "Design and Implementation of Burst Buffer Over-Subscription Scheme for HPC Storage Systems," in *IEEE Access*, vol. 11, pp. 3386-3401, 2023, doi: 10.1109/ACCESS.2022.3233829. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7266836

[77] Fredj, E., Laufer, M. (2023). Hyper Burst Buffer: A Lightweight Burst Buffer I/O Library for High Performance Computing Applications. In: Arai, K. (eds) Intelligent Computing. SAI 2023. Lecture Notes in Networks and Systems, vol 739. Springer, Cham. https://doi.org/10.1007/978-3-031-37963-5_23

[78] BeeGFS. Beeond: Burst buffer on demand. https://www.beegfs.io/wiki/BeeOND (2018).

[79] WekaFS Architecture: https://www.weka.io/resources/white-paper/wekaio-architectural-whitepaper/#wekaarchitecture

processes or can be deployed in containers. The system can bypass the kernel for many functions and by embracing the many-file-system paradigm it neatly avoids scalability issues of multiplicity of users and groups.The WekaFS driver supports up to 1024 file systems, and up to 24,000 snapshots in a single cluster.
WEKA storage servers are created by installing WekaFS on any standard AMD EPYC or Intel Xeon™ Scalable Processor-based hardware with sufficient capability, but requires SSD or NVMe solid-state drives. It does, however, consist of two separate layers, an NVMe SSD-based flash layer that provides high-performance file services to the applications, and an optional S3-compatible object storage layer that manages the long-term data lake.
In our testing Weka has proven to be a reliable fast-storage and scalable long term storage solution. The limiting factors were, predictably, non-free kernel drivers that often lag behind the last version and considerable price of investment due to pricing and hardware requirements.

A number of testing runs have been performed at JSI testing, BSC, and IT4I in internal testing deployments, courtesy of Weka. Reliability and performance has been in accordance with very high expectations and we found no issues with access standard implementations. We plan to publish results only from official deployments with Weka-approved hardware that we can not offer at this time.

7.2 S3 on Ceph storage

On EuroHPC Vega CEPH S3 storage[80] is available and can be used to transfer files to the cluster. Users can obtain authorization credentials for the storage using the OpenStack client. To interact with the storage, different S3 clients can be used and are well supported. We have carried out a proof of concept with bidirectional data transfers to/from the central storage node to stress test the protocol, optimize the set-up and test useability. S3 storage solution has been used in a deployment of a local data repository and in some tests for the InterTwin remote data orchestration tests. Further testing is planned, including transfer to remote S3 clusters.

8 Check-Pointing and Resilience

Resilience is one of the key challenges in maintaining high efficiency of future extreme scale supercomputers. With the increasing size of HPC computations, faults are becoming more and more relevant in the HPC field. The MPI standard does not define the application behaviour

[80] S3 storage on Vega: https://en-vegadocs.vega.izum.si/dcache/#storing-data-on-remote-dcache

after a fault, leaving the burden of fault management to the user, who usually resorts to checkpoint and restart mechanisms. This trend is especially true in highly interdependent applications, such as stencil applications (where each point's new value in the calculation depends on its neighboring points' values in the matrix), as their regular pattern simplifies the selection of checkpoint locations.[81] Since checkpoint and restart mechanisms introduce non-negligible overhead, disk load, and scalability concerns, checkpointing efficiency, when used as a resilience method in HPC, highly depends on the choice of the checkpoint interval.[82]

Standard analytical approach is to optimize the checkpointing interval for big, long-running jobs that fail with high probability and where the time lost can be very high, but this approach is unable to minimize checkpointing overheads for jobs with a low or medium probability ($0.2 < p_{fail} < 0.7$.) of failing. Analyses (c.f. ibid.) of batch traces of a number of HPC systems show that such jobs are, in fact, extremely common and represent a considerable loss of overall system efficiency. Multiple studies claim that the newer HPC systems are typically expected to be much less reliable due to increasing complexity of the system design, decreasing device dependability, shrinking process technology and increasing system size. Furthermore, multiple researchers have shown that the failure rates remain fairly stable during the stable operational period.[83,84] One overview study analysed the reliability characteristics of multiple large-scale HPC production systems over 8 years and included more than one billion compute node hours across five different systems.[85]

---

[81] Rocco, R., Boella, E., Gregori, D. and Palermo, G., 2024. To Repair or Not to Repair: Assessing Fault Resilience in MPI Stencil Applications. *arXiv preprint arXiv:2410.08647* https://arxiv.org/pdf/2410.08647

[82] A. Frank, M. Baumgartner, R. Salkhordeh and A. Brinkmann, "Improving checkpointing intervals by considering individual job failure probabilities," *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Portland, OR, USA, 2021, pp. 299-309, doi: 10.1109/IPDPS49936.2021.00038. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9460459

[83] Cappello F, Geist A, Gropp B, Kale L, Kramer B, Snir M. Toward Exascale Resilience. *The International Journal of High Performance Computing Applications*. 2009;23(4):374-388. doi:10.1177/1094342009347767 https://journals.sagepub.com/doi/abs/10.1177/1094342009347767

[84] Gupta, S., Tiwari, D., Jantzi, C., Rogers, J. and Maxwell, D. "Understanding and Exploiting Spatial Properties of System Failures on Extreme-Scale HPC Systems," *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Rio de Janeiro, Brazil, 2015, pp. 37-44, doi: 10.1109/DSN.2015.52. https://ieeexplore.ieee.org/abstract/document/7266836

[85] Gupta, S., Patel, Ti., Engelmann, Chr and Tiwari, . 2017. Failures in large scale systems: long-term measurement, analysis, and implications. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17). Association for Computing Machinery, New York, NY, USA, Article 44, 1–12. https://doi.org/10.1145/3126908.3126937 https://dl.acm.org/doi/pdf/10.1145/3126908.3126937

Check-pointing and resilience therefore need to become a central strategy to ensure scalability of application not just from the point of view of system administrators, where the system utilization is in focus, but also from the point of view of application developers, code designers and scientific users. Time to science and efficiency of allocation use become key parameters, but the result is perceived, in the context of large problems on large systems, as usability and scalability performance of the application. Resilience and checkpointing in view of the new architecture and possibilities of code supporting deployments on newer machines is therefore considered one of the central efforts of the MAX project, linking code development, testing and co-development.

8.1 Checkpointing under Slurm

Due to the wide proliferation of Slurm as the scheduler of EuroHPC machines, it is worthwhile to consider checkpointing and resuming under Slurm as the preferred solution. There are different solutions, including CRIU, set-ups with container-based checkpointing, the Berkeley Lab Checkpoint/Restart (BLCR) for Linux (requires a kernel component, checkpointable MPI libraries and application recompile) etc. Currently, the most development, deployments and testing involves DMTCP, its SLurm plugin and a number of addons and extensions, such as MANA, so we concentrate experiments on these solutions.

8.2 DMTCP

DMCTP, Distributed MultiThreaded Checkpointing[86] is a transparent checkpoint and restart tool that can preserve the state of an arbitrary threaded or distributed application to disk for the purpose of resuming it at a later time or in a different location. It requires no modifications to the application code, no changes to the system kernel and no special system privileges, operating directly on the user's binary executable. DMCTP can be operated by users without root access, making it usable in multi-user and multi-tenant environments.

DMTCP supports applications by supporting communication libraries, base libraries or language implementations. Among the targets are MPI (various implementations), OpenMP, MATLAB, Python, Perl, R, and many programming languages, shell scripting languages GNU screen sessions, including editor-based ones (vim/cscope, emacs) and it can checkpoint and restart X Window applications by supporting via TightVNC.

[86] https://dmtcp.sourceforge.io/ , https://github.com/dmtcp/dmtcp

DMTCP aims to provide support for checkpointing and restarting applications in HPC environments. With this goal, it supports the commonly used OFED API for InfiniBand, as well as its integration with various implementations of MPI, and resource managers, in particular SLURM.

DMTCP uses a central coordinator process to accept user instructions and manage C/R operation as shown in the figure below. There is one DMTCP coordinator for each application to be checkpointed. Application binaries are started under the `dmtcp_launch` command, causing them to connect to the coordinator upon startup. As threads are spawned, child processes are forked, remote processes are spawned via ssh, libraries are dynamically loaded, DMTCP transparently and automatically tracks them.

To support DMTCP integration with Slurm, a plug-in enables users to submit Jobs using Slurm so that the system automatically writes out checkpoint files and can restart those jobs from a taken checkpoint. This is usually enabled using a launch script. To restart a job, previously submitted via the launch script, the user can simply submit the restart script via `sbatch`.

While the checkpoint-restart support has improved considerably, there are many open issues related to MPI, GPU, hardware support, and specific libraries. Testing and support on specific machines is very important in order to achieve stable experience. Additional tooling available to overcome some of these issues is described below.

8.3 MANA

MANA[87] is an MPI-Agnostic Network-Agnostic transparent checkpointing tool. MANA employs a novel approach called *split-process* and is implemented as a DMTCP plugin, sharing its ability to operate without root access or kernel modification. In order to implement network-agnostic checkpointing, the process memory allocation is split into the application part and the MPI library part with libc and a MPI proxy library. Only the application part is subject to checkpointing, whereas the MPI part is re-initialized at restart, making the scheme network agnostic.

[87] https://github.com/mpickpt/mana

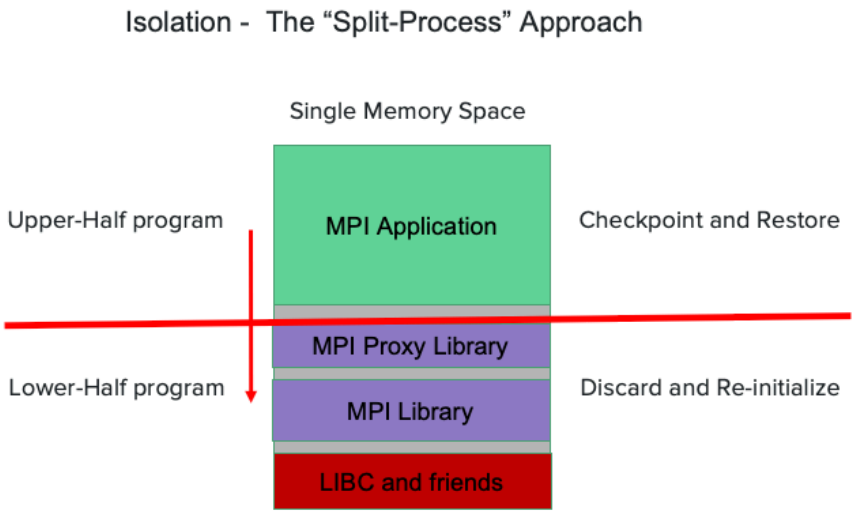

Figure 7. MANA sets-up a single system process that contains two programs in its memory address space, the lower half, containing an MPI proxy application with MPI library, libc and other system libraries, and the upper half, containing the original MPI application and data.

In order for this set-up to be operational, MANA has to wait for all MPI messages to drain before checkpointing. To support this, all MPI calls are prefixed with a trivial barrier. During the real collective calls the checkpointing is disabled, assuring that no messages floating in the network when checkpointing is initiated. MANA has been demonstrated,[88] since version 2, to enable transparent checkpoint-restart over thousands of MPI processes.

[88] Xu, Y., Zhao, Z., Garg, R., Khetawat, H., Hartman–Baker, R. and Cooperman, G., 2021, November. MANA-2.0: A future-proof design for transparent checkpointing of MPI at scale. In *2021 SC Workshops Supplementary Proceedings (SCWS)* (pp. 68-78). IEEE. DOI 10.1109/SCWS55283.2021.00019 https://par.nsf.gov/servlets/purl/10319913

CONCLUSION

We have presented an overview of emerging tools and techniques relevant to exascale workflows. Our study of a number of different profilers and performance analysis tools has been performed with the aim of choosing the selected tools that can help us improve performance, scalability, and efficiency of running MaX codes on the EuroHPC machines. This selection will inform our next steps, but we will remain open to update our tool set with new options since the area is under fast development. The wider study and its updates will be also of use to us when specific problems will have to be addressed and a more specialized tool will be made available for such a task.

For profiling our materials science applications, we will proceed with **Score-P** in combination with the **Scalasca Trace Tool**. This combination is ideal for large-scale applications and is highly effective in identifying performance bottlenecks. Score-P provides comprehensive performance measurement capabilities, while Scalasca excels in detailed trace analysis, making them a powerful duo for optimizing our application's performance. For memory profiling we will combine these tools with **Valgrind**. Valgrind is well understood and documented and while it offers simple generic performance analysis, it can be used with different executables without the need to recompile them and is therefore suitable for analysis as well as for standard continuous integration testing. We will try to identify which data structures in MaX codes are responsible for the largest memory usage. For I/O performance, Darshan will be tested in view of potential use as a monitoring tool in our CI/CD pipelines.
For testing in advanced scheduling, we plan to further evaluate Flux, potentially in combination with Kubernetes. This setup is expected to offer significant benefits for various use cases, providing a flexible and scalable solution for managing workloads. Flux's advanced scheduling capabilities, combined with Kubernetes' orchestration features, will help us achieve efficient resource utilization and improved job scheduling. We will also try to find solutions for AiiDA deployment with flexibility that Flux/Kubernetes can bring into the HPC environment.
We have concentrated on open source tools, because these tools can be used on all clusters, not being dependent on what is provided on the cluster and what is available from vendors. Availability on many systems is important also in order to provide comparable results and horizontal studies. This does not preclude the use of vendor-supplied tools, of course, and especially where such tools can offer deeper understanding of advanced systems.

Current testing of resilience and checkpointing solutions have revealed an increase of complexity: containers, reliance on accelerators and developments of interconnect require new and better integrated solutions. Regardless of this complexity, existing solutions are nearing the level where a robust solution for most MAX codes will be possible. However, an application workflow supported checkpointing solution with reliable re-starting and monitoring managed by the workflow manager remains the most robust and efficient solution for larger payloads and extreme scalability. The work on mini apps, modularization and optimization of MAX codes seems to support the possibility of further support of reliable workflow restarting.

In terms of distributed storage, we will continue with experiments in order to develop a proof of concept for reliable highly intensive data interaction for all MaX codes. Further testing of BeeGFS and Weka solutions within suitably equipped HPC set-ups will be performed. We aim to continue with tests S3 storage in a production environment. This choice is driven by the need for a reliable and scalable storage solution that does not rely on the Hosting Entities and their readiness to make configuration changes. S3 storage offers robust performance and ease of integration, making it a suitable option for our distributed storage needs. This decision will be reviewed based on design decisions in the EuroHPC ecosystem and development of exascale workflows within MAX regarding the data requirements.

Outputs: (1) Emerging technologies relevant for exascale workflows identified and tested; (2) A proof of concept for fault tolerance in case of unexpected termination of a task; (3) A proof of concept for reliable highly intensive data interaction; (4) Indications for adoption in the workflow infrastructure shared with the relevant developer teams.