H2020-FETHPC-01-2016

DEEP-EST

DEEP Extreme Scale Technologies

Grant Agreement Number: 754304

D5.1

Collection of software requirements

Approved

Version:	2.0
Author(s):	N. Eicker (JUELICH), Th. Moschny (ParTec), C. Clauss (ParTec)
Contributor(s):	M. Ott (BAdW-LRZ), D. Tafani (BAdW-LRZ), J. Corbalan (BSC), A. Jokanovic (BSC), V. Beltran (BSC), S. Krempel (ParTec), M. Nuessle (EXTOLL)
Date:	26.09.2018

Project and Deliverable Information Sheet

DEEP-EST Project	**Project ref. No.:**	754304
	Project Title:	DEEP Extreme Scale Technologies
	Project Web Site:	http://www.deep-projects.eu/
	Deliverable ID:	D5.1
	Deliverable Nature:	Report
	Deliverable Level: PU*	**Contractual Date of Delivery:** 31/December/2017
		Actual Date of Delivery: 31/December/2017
	EC Project Officer:	Juan Pelegrin

*– The dissemination levels are indicated as follows: **PU** - Public, **PP** - Restricted to other participants (including the Commissions Services), **RE** - Restricted to a group specified by the consortium (including the Commission Services), **CO** - Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

Document	**Title:** Collection of software requirements	
	ID: D5.1	
	Version: 2.0	**Status:** Approved
	Available at: http://www.deep-projects.eu/	
	Software Tool: LaTeX	
	File(s): DEEP-EST_D5.1_Software_Requirements.pdf	
Authorship	**Written by:**	N. Eicker (JUELICH), Th. Moschny (ParTec), C. Clauss (ParTec)
	Contributors:	M. Ott (BAdW-LRZ), D. Tafani (BAdW-LRZ), J. Corbalan (BSC), A. Jokanovic (BSC), V. Beltran (BSC), S. Krempel (ParTec), M. Nuessle (EXTOLL)
	Reviewed by:	N. Eicker (JUELICH) J. Corbalan (BSC)
	Approved by:	BoP/PMT

Document Status Sheet

Version	Date	Status	Comments
1.0	31.12.2017	Final version	
2.0	26.09.2018	EC Approved	

Document Keywords

Keywords:	DEEP-EST, HPC, Exascale, Software requirements

Copyright notice:

© 2017-2020 DEEP-EST Consortium Partners. All rights reserved. This document is a project document of the DEEP-EST Project. All contents are reserved by default and may not be disclosed to third parties without written consent of the DEEP-EST partners, except as mandated by the European Commission contract 754304 for reviewing and dissemination purposes.

All trademarks and other rights on third party products mentioned in this document are acknowledged as own by the respective holders.

Table of Contents

List of Figures

Executive Summary

The main objective of WP5 is to transform application requirements and thereof derived hardware characteristics into an MSA-aware system software architecture and programming environment. Therefore, especially its initial Task 5.1 plays a crucial role for the overall software design of DEEP-EST because here is where all the requirements for the system software architecture are initially gathered from the other work packages. In a second step these requirements will then serve to define the programming environment (by WP6) and the system software layers enabling management, I/O, and resiliency of the DEEP-EST prototype.

The deliverable at hand (D5.1) focusses at the first step and hence collects the requirements of the different applications together with their expectations towards the envisioned hardware architecture. In doing so, the document initially analyses all these requirements in terms of functionalities and interfaces needed by application codes, benchmarks and modelling tools with respect to the five task-related topics of interconnect management, network bridging, resource management, job scheduling and system monitoring. Furthermore, for each of these topics, a preliminary draft for a matching system software specification is derived on the basis of this analysis. An additional discussion of related requirements and implications on the programming environment to be defined in WP6 finally completes this document.

1 Introduction

The main objective of WP5 is the definition and implementation of the system software required to operate a Modular Supercomputer. For this, the initial task Tk 5.1 undertakes a detailed analysis of the requirements provided by the guiding applications handled in WP1 as the crucial input to the co-design efforts concerning the system software in the DEEP-EST project. In this context the applications influence the system software via the co-design in at least two ways: Firstly, applications establish direct requirements on the system software, especially on the Resource Management and the Job Scheduler. Secondly, an indirect influence is created via the requirements on the actual hardware of the Modular Supercomputer Architecture (MSA). It is obvious that design decisions drawn by WP3 e.g. on the interconnect architecture of the MSA will affect the work on the Interconnect Management or the Inter-Module Network Bridging undertaken in the context of WP5. Similarly, both direct and indirect constraints are put on the efforts of providing a programming environment for a MSA as conducted by WP6.

In order to identify such constraints a detailed questionnaire was created in a collaborative effort between WP3, WP5 and WP6 towards the application developers organised in WP1. This questionnaire and the detailed answers provided by WP1 are the main result of the early co-design efforts of the DEEP-EST project in order to define the Modular Supercomputer prototype to be developed within the course of the project.

This document summarises the findings and requirements extracted from the detailed answers provided by WP1 to questionnaire. It covers both, direct and indirect constraints on the system software to be designed and implemented in WP5 as well as on the programming environment to be provided by WP6. These requirements are accompanied by draft specifications of the software to be designed and implemented in both work-packages.

The document is organised along the line of the different implementation tasks as foreseen by the description of actions (DoA) for WP5. For each of these tasks a brief description of the context and state of the art is provided before the actual requirements as derived from the questionnaire are presented. Based on these findings, for each task a draft of the design and specification of the system software products to be developed is sketched and, wherever applicable, implications on the programming environment are discussed.

2 Interconnect Management

This chapter describes the interconnect management software architecture especially for the European interconnection technology fabri^3, which will be built in Work Package 4.

fabri^3 is a new variant of packaging of the direct EXTOLL network. Instead of adding a PCIe add-on board to each node (machine) of the network, a chassis with up to 64 EXTOLL Tourmalet ASICs is used to build the network. Within a single fabri^3 chassis, the Tourmalet ASICs are connected to the (compute) nodes by means of PCIe cables. Internally, a 3-D mesh structure is used to connect the Tourmalet ASICs with each other. External cable connectors are provided to either connect the fabri^3 chassis to further fabri^3 chassis (to scale to larger size network) or to close the dimensions of the 3-D mesh, forming a torus topology.

Interconnect management in this context includes the efficient handling of topologies, routing, and handling of faulty links and nodes. As fabri^3 in conjunction with routing nodes or gateways is expected to play a major role in the project's MSA, software has to be aware of cross-network traffic and the routing used for this traffic.

2.1 Context and State of the Art

There are a number of requirements that the interconnect management software for DEEP-EST has to fulfil. The basic functionality is already present in the existing EXTOLL management software as used in the DEEP-ER project. However, for increased reliability, fault-tolerance and global usage of the network, a number of additional requirements arise. Also, the newly developed fabri^3 hardware will be addressed by the interconnect management software. An EXTOLL network, such as the one employed in DEEP and DEEP-ER, as well as the fabri^3 in DEEP-EST, requires the calculation and distribution of the routing tables to the individual network nodes (in this case formed by EXTOLL Tourmalet ASICs) for the network to become operational. Changes have to be distributed to the individual nodes if the topology changes (e.g. because of a failure or because nodes or links are being added or removed). This is performed by a software packet named EMP (EXTOLL Management Process). This is in some ways similar to the task of the Subnet Manager in InfiniBand-based networks.

EMP currently is built with two major software components: the EMP master daemon and the EMP slave daemons. The master daemon process is executed on one of the nodes of the network. Each machine in the network forms a node of the network and features a Tourmalet ASIC attached to it via PCIe. Management is multiplexed on the same physical connection (PCIe) to the ASIC as the actual network traffic (i.e. MPI traffic). PCIe offers a high-speed configuration and management path to the network hardware, but this also creates a single point of failure as well as it may add some problems when high-speed traffic and EMP access the network concurrently, especially in presence of faults. In the existing EMP, the other nodes of the network are configured remotely over the EXTOLL network itself, which again offers a high-speed, flexible path to access the hardware from EMP, but is also prone to network faults on certain conditions, for example if the network path that leads from the management node to the fault suffers from a fault itself. A second component of EMP is the EMP slave daemon,

which may run on each machine in the network and communicates via TCP/IP with the master daemon. It is mainly used today to implement network monitoring.

The existing EMP software already offers a number of important features for network management:

- network topology discovery
- topology specification using a DSL, including support of hierarchical and inter-connected (sub-)topologies to specify complex networks
- routing calculation, including multi-cast routing, both for simple and hierarchical topologies
- monitoring capabilities of basic networking parameters
- ability to stop and resume network traffic globally, to enable scheduled hardware interventions
- REST-based API interface
- simple CLI-based interface
- web-based front end

Routing is certainly one of the basic and most important requirements to be provided by interconnection management software. Routing should be first of all correct and dead lock-free. It should also facilitate high-performance, preferably minimal routing. The current EMP version only implements deterministic routing on the Tourmalet chips, although the hardware also supports adaptive routing. As stated above, EMP can already handle “hierarchical topologies”. This feature has also been used extensively in the scope of the DEEP-ER project. An example is the existing DEEP-ER SDV machine, which actually consists of three parts:

- 16 general-purpose compute nodes based on Intel Xeon processors interconnected in a 4-D Hypercube topology
- 3 general-purpose storage nodes, interconnect in a 1-D ring topology (equivalent to a 1-D torus)
- 8 KNL based Xeon Phi nodes, interconnected in a 2-D torus topology (4×2 nodes).

All of the three parts are connected by a number of direct links from one part to each of the other parts. As such, an all-to-all topology between these three parts is formed. So, a hierarchy of topologies is formed which describes the topology of the complete machine.

Multicast routing is implemented by the current version of EMP and is already used, for example in the DEEP-ER context to transport multicast/broadcast packets of EXN, the EXTOLL Linux Ethernet emulation layer, offering TCP/IP transport services over EXTOLL. This transport service in turn was for example used to carry the BeeGeeFS storage traffic in DEEP-ER. Multicast hardware support can of course also be used to carry other multi-cast traffic, as for example in the case of certain MPI collectives.

2.2 Requirements for System Software

The interconnect management software for DEEP-EST has to support the new network implementation called fabri^3 used within the project. This implies certain new requirements and possibilities. Most importantly, the EMP daemon and slave daemons shall no longer run on actual compute nodes of the machine but instead on the management CPU provided and integrated into the fabri^3 hardware on the chassis level. With the EMP daemon running independently of all individual nodes, there is no problem anymore if exactly the machine that runs the EMP master daemon fails or has to be rebooted. The new hardware structure also divides the power domains of the compute machines and the network nodes. This means, if a machine loses power or has to be power cycled or replaced, this does not automatically break the topology of the network, since the power to the corresponding Tourmalet ASIC can be retained. Obviously, EMP in DEEP-EST shall take advantage of these new possibilities to increase RAS. Furthermore, in fabri^3 the management software can access the network hardware using a side-band channel completely independent of the high-speed communication channels actually used by MPI, storage, etc. This is accomplished on the hardware side by a hierarchical connection of every of the 64 Tourmalet chips of a single fabri^3 chassis to the management CPU within the chassis. The actual hardware interface of Tourmalet used here is an I^2C interface. While this interface is considerably slower then PCIe, the complete independence from normal hosts and host traffic far outweighs this restriction in speed. Also, the management CPU can access each of the Tourmalets using the I^2C-based access, no remote access over the actual EXTOLL transport is necessary anymore. Both of these possibilities of the hardware shall be translated into requirements in the sense that EMP leverages them to further increase the availability of the network in cases of faults.

Since multiple fabri^3 have multiple management CPUs that will be connected by a management Ethernet, it is necessary to allow them to interact again using a completely independent side-band channel. In this sense the software shall take into account scaling of the network to multiple fabri^3 chassis interconnected to each other, forming a larger topology. Of course, the possibility to form hierarchical topologies shall be retained and actually be improved upon.

Interconnect management in DEEP-EST shall also automatically detect and handle faults in the network wherever possible. If necessary, semi-automatic handling shall be implemented as a fallback if fully automatic resolution is not possible for a certain condition. Specifically, management software shall be able to detect and correctly handle single link failures in an automated fashion. Of course, some (transient) performance implications may occur, as the link is removed from the network and the network reconfigured. If possible, packet loss shall be avoided in this case. It remains to be analysed whether this will be possible in all cases. Also, compute node machine failures (i.e. PCIe link goes down) will be detected. A protocol will be implemented to handle this failure in a deterministic fashion involving reconfiguration of the network to make sure no global impact is seen, i.e. the network remains operational. Of course, packets destined to this node will be lost, respectively package delivery will not be possible anymore. When a node comes up again, it shall also be added to the network again automatically.

On the routing side, a number of improvements will also be realised. This includes an analysis of the current routing algorithms, and whether any additional ones are necessary. Currently EMP supports dimension-order routing, torus routing, shortest path first, Up/down and

Up*/down* routing for the deterministic part, and no adaptive routing. At least adaptive routing shall be added.

Also on the routing side, preparations will be taken to enable fast re-routing of traffic in case of local failures. See below for some more details on this.

It remains to be seen whether network federation will add to the routing requirements of EMP. Currently it is believed that no special routing strategies have to be added for this.

As there will also be work done on implementing efficient offloading of collective operations to the network using the GCE, an interface to query topology information from EMP will be implemented that can be leveraged by middleware libraries, like MPI.

2.3 Draft for Design and Specification

With respect to the base EMP implementation, the existing software stack has to be ported and adapted to run on the fabri^3 management CPUs. Most probably these will be standard x86-based CPUs running a standard Linux distribution, which of course will ease this process. Even in the case a different CPU will be chosen, porting of core EMP should be straightforward, since it is implemented in Scala on top of a Java Virtual Machine, enabling good portability. But in any case, EMP has to be adapted to the different access path to the hardware. Instead of accessing the hardware over PCIe and memory-mapped I/O as well as remote register file accesses over the Remote Memory Access (RMA) unit of the EXTOLL network, hardware shall be accessed over a hierarchical bus system terminated at the I^{2}C interface of each Tourmalet chip. On the management CPU this could probably be based on one or multiple USB interfaces. As there will be more than one EMP master daemon running in a multi fabri^3 system, EMP will be extended to support a cooperative network configuration. In this new configuration scheme each master configures its own network partition (one fabri^3), whereas an elected super master configures the inter-partition routing. Also, a new protocol for data exchange between master daemons will be implemented.

New, distributed protocols will be implemented for automatic handling of failing links and nodes. Note that there are combinations of multiple failures that software will not be able to resolve automatically. A pathological example of this is the case where multiple links fail in such a way that the network is actually divided into two distinct parts. The link failure protocol will involve reconfiguration of the routing in such a way, that packets are re-routed after a link failure has been detected. The difficult part here is to try to avoid doubling of packets, which is going to be evaluated within the implementation. If it is impossible to avoid this under all circumstances, a running application may see errors, but the global system shall remain operational. The case of a failing endpoint (compute machine) can be detected by the PCIe link not being available anymore. In this case, reconfiguration shall occur to make sure that the network will not come to an halt because of back pressure and the node will be removed from the set of possible destinations. Once the node comes back (or is initially seen), the node will be properly configured and added to the set of reachable sources/destinations again.

The REST API will be extended or a new API will be added to enable access to topology information for middleware libraries, for example to foster efficient collective implementations and interact with GCE software.

On the routing side, adaptive routing shall also be added. Adaptively routed packets are not guaranteed to arrive in-order at the destination. It is the responsibility of upper-level protocols to handle this correctly. This shall be no problem for TPC/IP over EXTOLL for example, and possibly also not for bulk-level MPI data which arrives as the response in a rendez-vous protocol. MPI requests will surely remain in the deterministic realm since in-order package processing is mandatory in the standard.

3 Inter-Module Network Bridging

On the one hand, when looking at Figure 1, it becomes clear that the designated *Network Federation* (NF) of the Modular Supercomputer Architecture (MSA) is intended to serve — when considered as an abstracted entity — as some kind of a “glue” that shall provide a high-speed connectivity *between* the different system modules. On the other hand, when considering the modular nature of the MSA and hence of the later DEEP-EST prototype, it becomes obvious that different network technologies might also be employed *across* the different modules of the system. So, for instance, different application demands might require different network characteristics to be satisfied by different technologies within the respective modules. Moreover, as the idea of extensibility is a vital aspect of the modular approach of DEEP-EST, also different generations of the same network technology might be employed within the various modules of a system that has evolved over time.

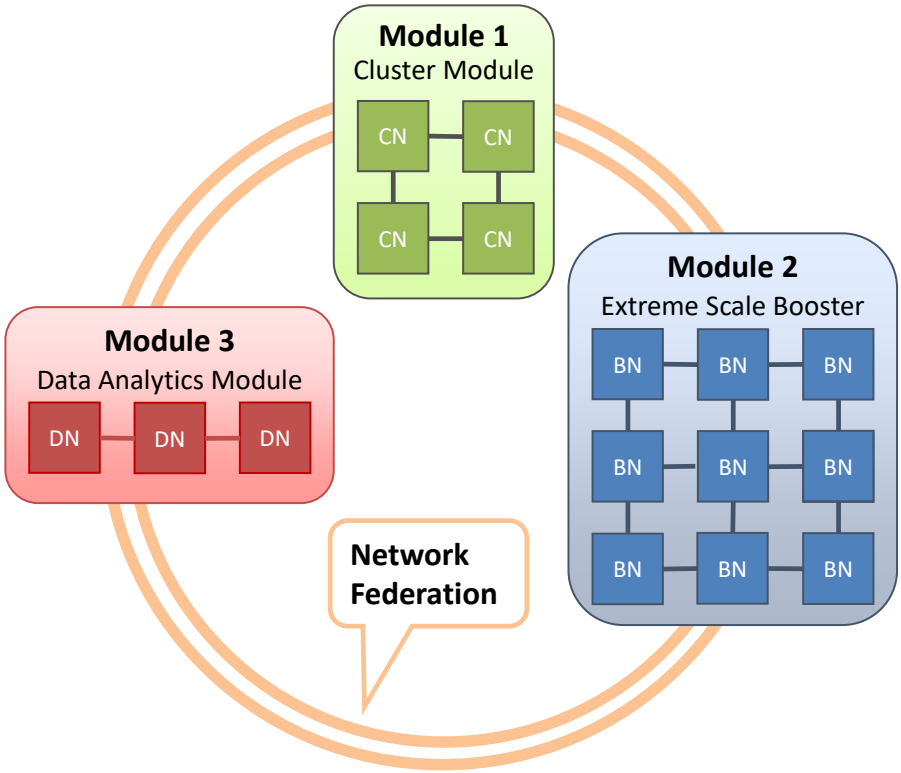
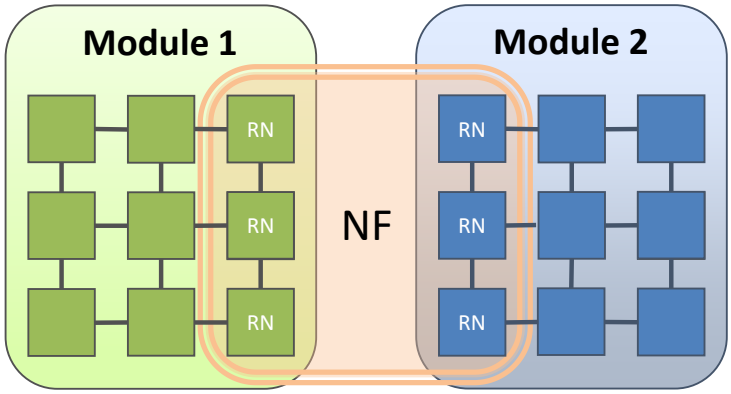


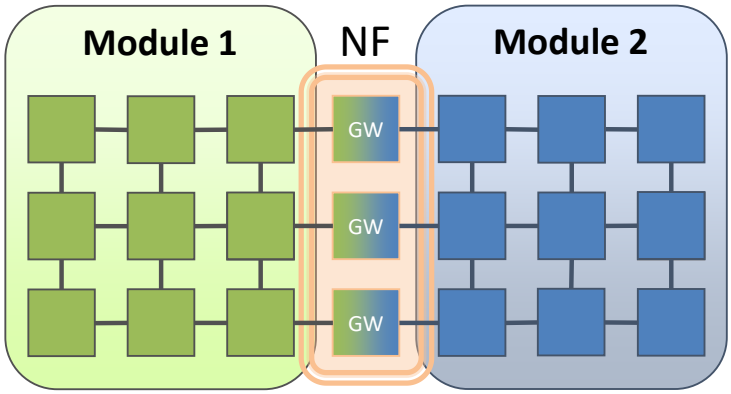
Figure 1: Logical view of the Modular Supercomputer Architecture

However, provided that the DEEP-EST prototype will feature different network technologies within different modules, the question remains which manifestation the Network Federation will exhibit. Although there has up to now no final decision been taken in this regard (initial, high-level decisions on the actual system architecture are due together with D3.1 in Month 6 and more detailed hardware-related specifications — most probably also with respect to the Network Federation — are not to be expected before the preparation phase for D3.2 with due in Month 12), only a few conceptual approaches [10] suggest themselves for employment here:

Either there will be in addition one dedicated global network with linkage to all modules (but not necessarily to all nodes of the modules, see Figure 2a), or additional and likewise dedicated gateway nodes with simultaneous access to two (or even more) network technologies have to bridge between the separate modules, as shown in Figure 2b.



(a) Network Federation based on an additional global network and Router Nodes (RN)



(b) Network Federation based on additional Gateway Nodes (GW) that link between modules

Figure 2: Different manifestations of the Network Federation

Certainly, in both cases, there will also be demand for an additional system software layer that acts as a mediator and forwarding instance between the different networking domains. In turn, this bridging layer will most probably be embodied by daemon processes running on these dedicated gateway or router nodes for mediating between the different protocols and interfaces. However, apart from the hardware configuration, the actual design of this infrastructure for network bridging will strongly depend on the software-related requirements of the applications concerning *inter-module communication*. By analysing those requirements, as they have been collected and presented in D1.1, we have identified two outstanding aspects of the network bridging that are to be stressed: These are namely the demand for low-latency/high-bandwidth MPI communication even across module borders; as well as the demand for globally accessible *Network Attached Memory* (NAM) regions — both to be provided by a bridging system software layer that shall be drafted in this section of D5.1.

3.1 Context and State of the Art

The starting point for the development of a bridging software will be the experiences from the DEEP project, where a very efficient and high-performance bridging solution between InfiniBand and EXTOLL has been developed on the basis of the ParaStation *pscom* communication library [11]. pscom is a low-level communication library specifically designed for utilisation in HPC systems. Although its main objective is to serve as the lower-level communication substrate of ParaStation MPI, it can also be used independently as a light-weight standalone communication library. In fact, this feature renders pscom again as the ideal candidate for being the communication solution for the envisioned gateway daemons also in DEEP-EST.

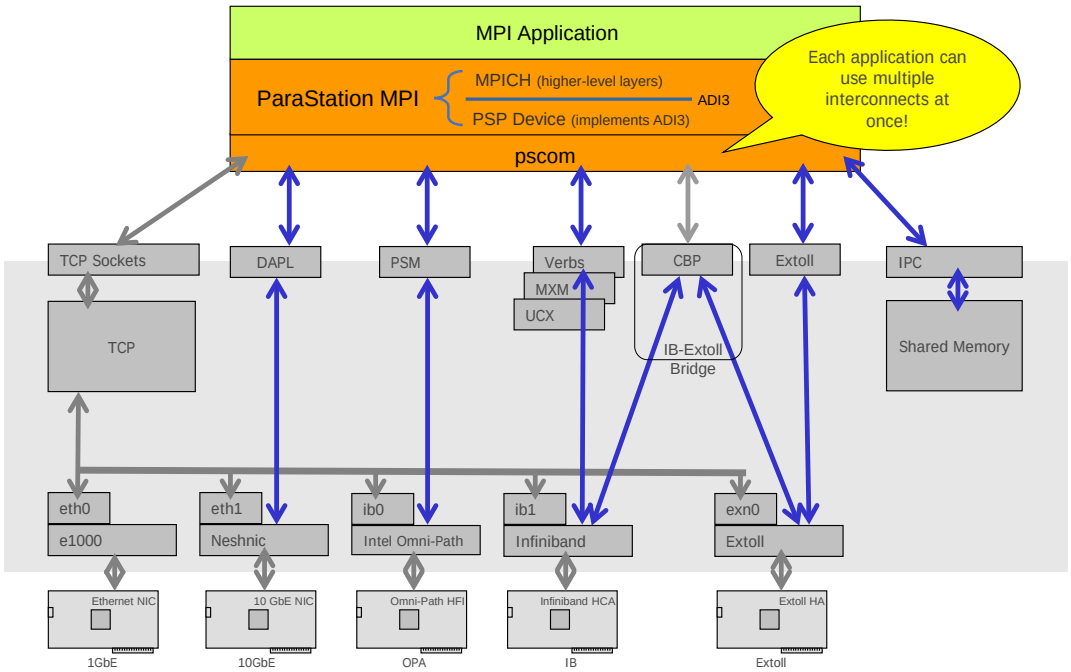


Figure 3: Software layers of ParaStation MPI and pscom with its various plugins

As pscom targets HPC systems, a variety of interconnects and interfaces commonly used in this specific domain are supported. In doing so, pscom exhibits a flexible architecture featuring *plugins* for all the different interfaces and protocols, as shown in Figure 3. These plugins are loaded and selected at runtime of the library by means of a priority/fall-back scheme, i.e., plugins promising faster communication than others are preferred while slower but probably more robust ones can still be used in case of an unsuccessful initialization of the former. Moreover, as pscom inherently exhibits the capability to support different network technologies even in parallel, this plugin selection feature can directly be used also for implementing the routing across gateways: According to this approach, whenever two processes cannot connect directly, they eventually select a designated *gateway plugin*, which then in turn chooses an appropriate gateway daemon and connects to it via the best locally available plugin, i.e. the local high-speed network. In that way an extended connection between both processes can be established through the gateway, for example, on the basis of a *store-and-forward* mechanism for message chunks to be passed along by the respective daemon.

A productive environment, where the *Cluster-Booster* concept being the predecessor of the MSA has recently been implemented in this way, is the JURECA system at the Jülich Supercomputing Centre. This system features an Intel Haswell-based general-purpose Cluster of 1882 nodes equipped with InfinBand (IB), as well as a Xeon Phi-based Booster of 1640 nodes with Omni-Path (OPA). Both subsystems are connected via 198 bridge nodes that are equipped with both InfiniBand and Omni-Path NICs. In that way, MPI traffic can be routed across these bridges either via TCP/IP (over IB and OPA) or by means of gateway daemons running on those bridge nodes and making direct use of the respective high-speed networking APIs (i.e. IB Verbs and PSM2, respectively) to forward the message chunks.

However, it has still to be evaluated if this approach is also the way to go in DEEP-EST. This is because, when looking at the outcome of the DEEP project, even more sophisticated approaches than a simple store-and-forward mechanism are also conceivable. So, for instance, the bridging solution between InfiniBand and EXTOLL as developed in DEEP leveraged a combination of *Remote Direct Memory Access* (RDMA) techniques provided by both networks to avoid unnecessary utilisation of the processors of the Booster Interface Cards (BIC) — which were the analogy to gateway nodes in the DEEP prototype. In doing so, the communication between the Cluster and the Booster was divided into two kinds of channels: Dedicated control channels towards the daemon processes running on the BICs were used to set-up the high-speed data channels based on RDMA between the MPI processes running on Cluster or Booster.

Depending on the features of the bridged networks, such a smart mapping approach based on RDMA capabilities might also be viable in DEEP-EST. Hot candidates in DEEP-EST for becoming part of the Network Federation are EXTOLL Tourmalet, Mellanox InfiniBand, and Intel Omni-Path. In addition, also Atos BULL BXI and even Gigabit Ethernet may come into consideration [12]. However, EXTOLL will definitely serve in DEEP-EST as the type of interconnect the Network Attached Memory (NAM) nodes will be attached to, since the NAM technology depends on EXTOLL and its RMA capabilities. If and how the other mentioned fabrics could integrate into this Network Federation has still to be investigated, but it is already clear that some technologies would have certain drawbacks. So, for instance, the Omni-Path architecture is known for being prone to burden the main CPU with communication-related workload instead of off-loading it to the network [13]. A motivation for this *on-load* approach of Omni-Path is the assumption that on multi- and manycore CPUs some cores are frequently idle and can thus appropriately be used for network operations. However, considering in particular the weak single-thread performance of current manycore processors, it is at least questionable whether such an on-load architecture would be the ideal candidate for being a backbone of the Network Federation [14] — all the more since the on-load feature virtually rules out the opportunity for RDMA-based mapping of different interconnect technologies, as it has been done in DEEP.

As mentioned above, NAM modules can be attached to any EXTOLL link in the Network Federation [15]. On the downside, as the NAM technology is based on the EXTOLL RMA engine, it can *not* be attached directly to any other network type than EXTOLL. However, within the EXTOLL fabric, each NAM is in principle directly accessible from any compute node by utilizing a dedicated software stack called *libNAM* that is based on the EXTOLL RMA API. In doing so, the programming interface of the libNAM allows applications to direct NAM-related memory allocation requests towards a dedicated *NAM Manager* that acts as a central instance for managing all NAMs within a fabric. The actual memory accesses towards the NAM memory regions

can then be performed by means of dedicated *Put* and *Get* operations provided by libNAM.

3.2 Requirements for System Software

First and foremost, we have identified the demand for high-performance MPI-based communication even between system modules as a very important requirement for the inter-module bridging. This is because, after analysing the outcome of the application questionnaire as it has been presented in detail in D1.1, there are several DEEP-EST prototype applications that will certainly distribute their workloads and will accordingly map their code parts concurrently to different modules of the system.

So, for instance, the application tool chain of *NEST* [2] (simulation of brain-scale neuronal networks) with its downstream applications *Arbor* [4] and *HybridLFPy* [3] (local field potentials predictions) as well as *Elephant* [6] (in-situ data analysis) will run its processing steps simultaneously within different modules by using MPI via the *MUSIC* [5] code coupling library for the inter-module communication (see Section 2.1 in D1.1). Although the maintainers of the NEST use case in DEEP-EST predict that the respective bandwidth and latency requirements for this inter-module communication will be limited — at least when coupled with Elephant — there is the affirmed demand for a non-congesting delivery of spike information, encapsulated in a compact data representation, to be transferred frequently from NEST, running on the HPC Cluster Module (CM), to Arbor/HybridLFPy, running on the Extreme Scale Booster (ESB), via MPI — which is a demand that is supposed to be satisfied by a latency-optimized inter-module communication path.

Moreover, other DEEP-EST applications, as for instance *GROMACS* [7], already exhibit an *intrinsic* modularity on the algorithm level. Therefore, it seems quite natural to map each of such distinct code parts of those single applications likewise onto different modules of the DEEP-EST system. However, MPI communication within a single application is often much more complex (e.g. in terms of frequency, load and patterns) than it is usually the case for multiple applications that are coupled via MPI as a workflow (e.g. typically downstream communication only in one direction). Therefore, when decomposing a single application by mapping it onto multiple system parts, appropriate measures have to be taken to prevent the inter-module communication to become a bottleneck. So, for instance, the maintainers of the GROMACS use case in DEEP-EST have stated a demand for low latencies and high bandwidths for good MPI scalability (see Section 3.2 in D1.1). Hence, when assuming that GROMACS is to be distributed across multiple modules of the DEEP-EST prototype, this statement will presumably also hold for the inter-module communication.

All in all, by looking at this from a more abstract point of view, we have identified the following three generic communication scenarios regarding the mapping of applications (or parts thereof) onto multiple modules of the DEEP-EST system:

Peer-to-peer-related communication with two (or more) code parts of a single application running in parallel on two (or more) modules and hence communicating intensively among each other. This scenario will most probably be latency-bound and it is quite likely that all of these code parts are to be started simultaneously on the different modules, e.g. by means of a single `mpiexec` call (cf. pp. 362 ff. in [1]).

Master/worker-related communication between two (or more) code parts where certain workloads are offloaded by the respective application to modules particularly suited for accelerating them. This scenario will probably be bandwidth-bound and it seems quite natural that the offloaded processes are to be started in a dynamic fashion, e.g. by means of `MPI_Comm_spawn` (cf. pp. 374 ff. in [1]), as it has also been utilized in DEEP.

Workflow-related communication between two (or more) steps of a data process chain where the different processing stages and their tools are executed subsequently on different modules in a pipelined fashion. Up to now, this scenario is commonly realized via storage I/O, where each subsequent step reads those datasets as input that have previously been generated and written to disk by the predecessor step.

In the context of DEEP-EST, we see — especially for the latter scenario — two new approaches coming up for the inter-step communication: The first one is to use the NAM as a very fast temporary storage for the intermediate data to be passed from one step to the next; the second approach is to use MPI-based communication for passing the data directly — for example, between two (or more) formerly independent MPI sessions that have connected to each other by means of `MPI_Comm_connect/accept` (cf. pp. 386 ff. in [1]).

However, the MPI-based approach would require that the related processing steps to overlap in terms of their MPI sessions for the data passing between them — which certainly demands for additional measures with respect to scheduling and resource management (see also Section 4). In contrast to this, the usage of NAM nodes that are capable of being addressed quite similar to MPI ranks by means of one-sided *Put* and *Get* operations could be exploited to decouple the different processing steps by avoiding an otherwise required session overlap.

In fact, several of the DEEP-EST prototype applications (in particular the Machine Learning uses cases *HPDBSCAN* [8], *TensorFlow* [9] and *piSVM*) indicate that they will seek to utilize the NAM as such a temporary storage for intermediate data between processing steps. However, since the envisioned workflows of these applications are in contrast to be distributed across *several modules* of the MSA, it is important to have fast access to the NAM from *everywhere* in the DEEP-EST prototype (see Section 8 in D1.1).

As one can see, this requirement for having *globally accessible NAM nodes* renders an obvious obstacle to the fact that only EXTOLL links can be attached with the NAM technology. As a consequence, either the whole Network Federation needs to be built on EXTOLL basis, or some low-level gateways on software basis would have to perform the required translations of network packages in order to mediate between the EXTOLL-based libNAM API and the foreign network technology, as it is depicted in Figure 4.

3.3 Draft for Design and Specification

With respect to MPI-based inter-module communication, we can at this point only sketch a quite generalized specification since the actual hardware features of the later DEEP-EST prototype — in particular regarding the manifestation of the designated Network Federation — are unfortunately still unknown in this early stage of the project. Therefore, we currently conceive a daemon-based approach, which will most likely be quite similar to the ones that have already implemented successfully in the DEEP project and/or for the JURECA system. However, since

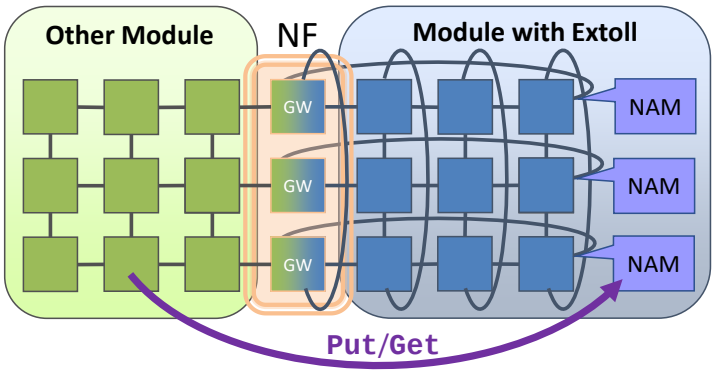


Figure 4: Globally accessible NAM nodes attached to a local EXTOLL network

it is a key objective of Task 5.3 to extend the number of supported combinations between interconnect technologies in DEEP-EST, a simple adaptation of these inherited solutions will not be sufficient. This is because each thinkable combination of network technologies will bring its own set of challenges and opportunities especially with respect to performance optimizations. Nevertheless, as performance optimizations do not have priority at this early stage, our current draft for design and specification naturally has to be rather generic here.

The envisioned MPI bridging solution will — from an abstract point of view — most likely look like the layered model shown in Figure 5. Here the already described feature of the pscom library to drive multiple of its plugins in parallel by selecting protocols and interfaces for communication on a per-rank basis (either based on priorities or on an explicit configuration) is highlighted.

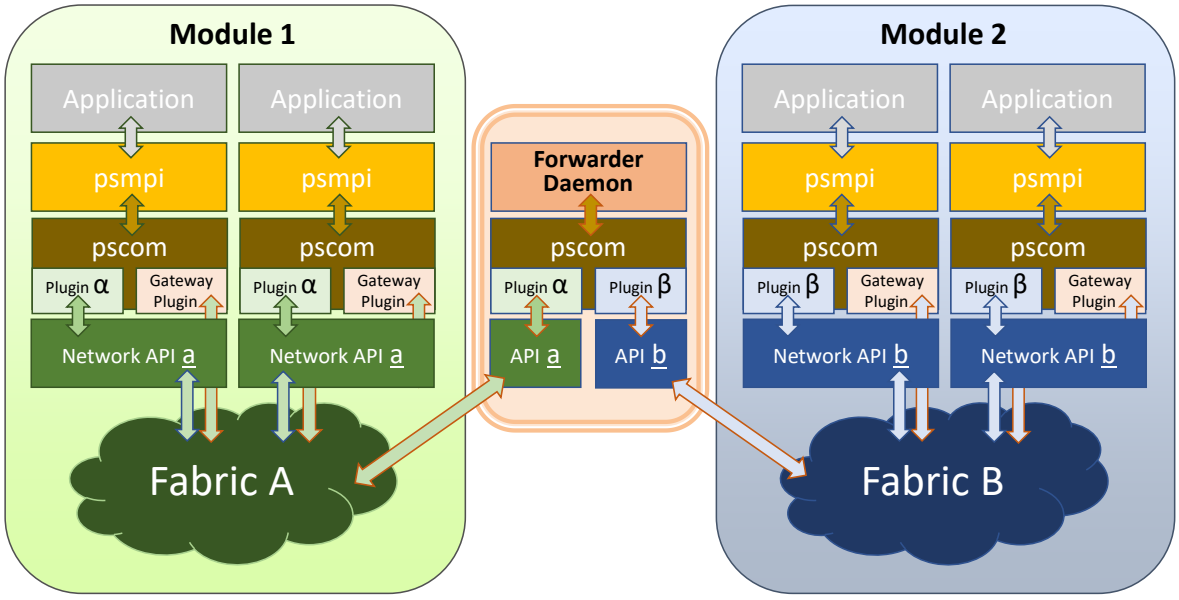


Figure 5: The envisioned daemon-based forwarder concept for the network bridging

The left hand side of this figure shows two compute nodes of an MSA module 1 that utilizes the pscom plugin α for intra-module communication via the native software interface a to the local high-speed network A. In contrast, on the right hand side of this figure, two further nodes of the MSA are depicted, which are part of a different module 2 and which therefore leverage a likewise different network technology B by means of a native software interface b being encapsulated by the respective pscom plugin β for the local communication. In addition, and for simplicity reasons not shown in this figure, processes located on the same node make use of the shared-memory plugin of pscom for the intra-node communication.

On all compute nodes of both modules, an additional *gateway plugin* is designated for the bridging communication with the forwarder daemons in case of inter-module messages. Internally this special plugin then again uses the module-native plugins — α on the left and β on the right side of Figure 5 — for the message transport towards the daemons. For doing so, each of the inter-module messages needs to be extended by additional information like MPI rank and network address concerning the final destination in the other fabric. The forwarder daemon then has to analyse and remove this information again, before sending the message within the other fabric towards the final receiver. Conversely, on the respective receiving compute node, the gateway plugin accepts the messages sent from the gateway daemon via the native transport for satisfying all matching receive requests as posted by the application with respect to MPI ranks within the other fabric.

As one can see, this architectural design is pretty much in accordance with the gateway approach as drafted in Figure 2b. In case it turns out that the router-based approach is the way to follow in consequence of a further tier in the network hierarchy (see Figure 2a), a likewise additional *daemon-to-daemon* communication step would need to be introduced. However, since this architectural extension could be implemented straight forward, and as we hence currently see no requirement for a fundamental design deviation for this, we believe that we can stick in all conscience to the design specification as detailed above in either case for the network bridging.

Concerning the global accessibility of NAM nodes, the interesting scenario for the DEEP-EST prototype configuration would be the heterogeneous case of a network federation where NAM nodes attached to EXTOLL links shall be accessible even from within modules that internally utilize a different network technology. This is because in particular this scenario will require a software-based solution for the NAM-related network bridging — whereas for a bridging between distinct EXTOLL (sub-)networks in different modules a hardware-based routing solution might probably be preferable.

Figure 6 illustrates how such a software-based solution, which is likewise in accordance with the gateway approach, could look like for inter-module NAM accesses: The right hand side of this figure shows a module equipped with an EXTOLL fabric that in addition features attached NAM nodes — which are consolidated in this layer diagram as a supplementary *NAM Module*. NAM accesses issued within the EXTOLL fabric can hence be transacted directly, e.g. by utilizing *Put* and *Get* operations as provided by the libNAM API, while NAM-related memory allocation requests are to be handled by the NAM Manager. As opposed to this, *Put* and *Get* requests towards the NAM that are issued from within the non-EXTOLL fabric on the left of this picture need to be forwarded by software daemons running on respective gateway nodes. To do so, the issuer within the non-EXTOLL module may use a dedicated wrapper library that mimics the libNAM API by embedding all NAM requests, for example, into control messages to be

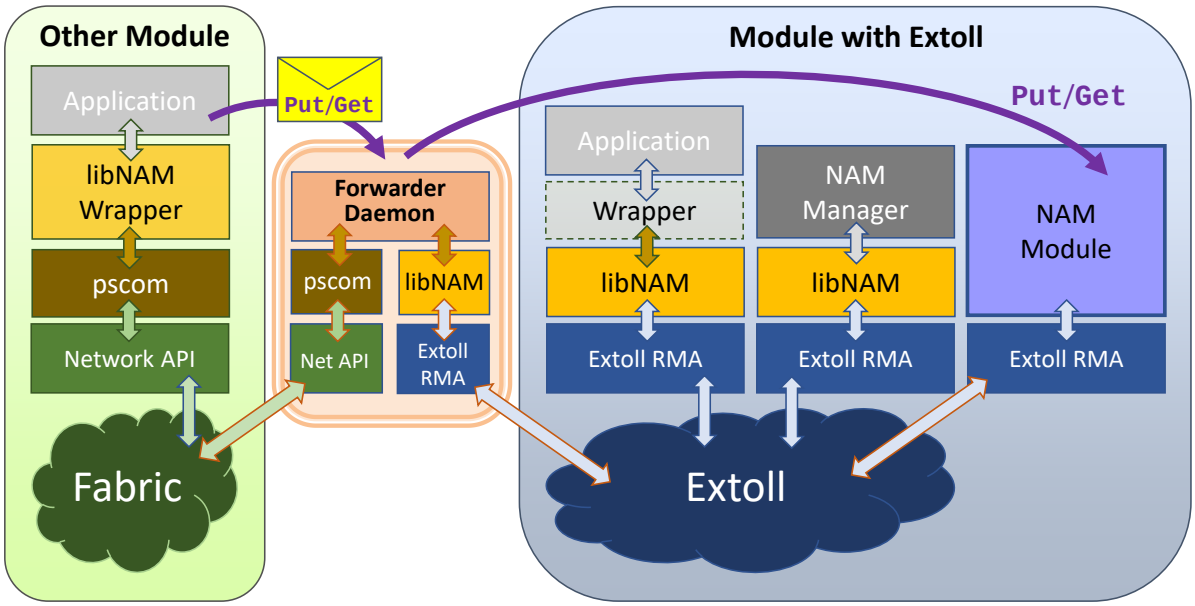


Figure 6: The envisioned software architecture for globally accessible NAM nodes

transferred via pscom connections. The gateway daemons can then receive such requests sent via pscom and translate them into actual libNAM operations to be forwarded via EXTOLL to the NAMs. That way, transparent access to the NAMs by means of a unified API (i.e. the libNAM wrapper interface on the left and the actual libNAM API on the right side) is possible from within both modules.

However, in addition it is also conceivable that both sides may utilize an even more generic programming interface than libNAM. For instance, we also envision the possibility to access the NAMs via MPI-based *Put* and *Get* functions which in turn make internal use of the libNAM — at least from within the EXTOLL fabric. According to this, the depicted wrapper layer beneath the application shown on the right side of the figure as well as the aforementioned libNAM wrapper on the left side would both actually represent such an extended MPI library.

4 Resource Management

4.1 Context and State of the Art

In this project, we consider the Resource Management to consist of two parts, the Resource Allocator and the Process Manager.

Resource Allocator

The Resource Allocator works hand in hand with the Job Scheduler focused on in Task 5.5 and described in Section 5. In general it has two main tasks. The first is to take requests from the users and pass them over to the Job Scheduler. The second is to take decisions of the Job Scheduler and trigger the Process Manager in time to prepare the correct resources and start the users' processes in the right way at the right resources.

In the DEEP-EST project, the Resource Management will be based on the orchestration between the SLURM Workload Manager [32] (Resource Allocator plus Job Scheduler) and the ParaStation Process Manager. SLURM is an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. It is actively developed by SchedMD [33] and widely used in the HPC community. It is highly configurable and customizable via plugins and there already exists a good integration with the ParaStation Management Daemon, the Process Manager of our choice.

From the SLURM perspective, a *job* is one scheduling unit and can consist of several *job steps*. With the job submission, users request a set of resources which are required to run the job. When the job is scheduled, an *allocation* with a fixed set of resources is created and handed over to the Process Manager to start the job steps inside. In vanilla SLURM, the Resource Allocator part is done by the SLURM Control Daemon (*slurmctld*) running on the head node of the cluster as a central instance. The Process Manager part is executed by the SLURM Daemons (slurmd) running on each node of the cluster providing resources. With the ParaStation Management Daemon as Process Manager, the SLURM Daemon is replaced by *psslurm* – a plugin to the ParaStation Management Daemon talking directly to the SLURM Control Daemon – taking over and enhancing all tasks of the original SLURM Daemon.

Process Manager

For jobs which are using resources distinct from each other, the work for the Process Manager is pretty straightforward. It just has to start the executables on the right cluster nodes and forward I/O to/from the nodes. This is already working very well today and has no bigger scaling issues. In this exascale-focused project we are looking on the other, much bigger part of jobs running on cluster systems, those needing a communication infrastructure between the processes running on different nodes, most commonly realized using the Message Passing Interface (MPI).

The general role of the Process Manager as the actual MPI runtime environment can commonly be characterized by the following four tasks:

Process Launcher The Process Manager is responsible for receiving the requests for parallel MPI sessions as issued, for example, via the `mpiexec` command, to check for free resources in terms of unpopulated cores within the allocation, and to spawn the parallel processes onto these cores while applying an appropriate pinning scheme. Allocation-internal scheduling policies are also subject to this launching procedure.

Connection Broker Upon creation of the processes, all required network connections between them have to be established. This is the key prerequisite to setup a true MPI session where each process can directly communicate to all the others. For this purpose, it is usually necessary to gather and then to distribute certain information about the communication endpoints (e.g. IP addresses and port numbers the processes are listening to). In many MPI implementations it is the responsibility of the Process Manager to facilitate this initial information exchange by means of some out-of-band messaging system.

Process Controller During the runtime of a session the parallel processes are usually kept under control of the Process Manager so that it can instantly detect any process fault and then take appropriate measures for gracefully cleaning-up the whole session in such a case. By this means a proper reuse of resources can be ensured after session termination.

I/O Forwarder Especially for interactive sessions users demand that standard output appears on the console where they launched the respective session, and the same applies all the more to the reverse direction where user input is to be forwarded from their local console to the processes of the session. As the host for the console and the nodes the session is running on commonly do not belong to the same operating system domain, it is usually the responsibility of the Process Manager to forward standard I/O between the user and the distributed processes of a parallel session.

All major MPI implementations come with a Process Manager accomplishing the tasks above. However, although the `mpiexec` command is standardized, its implementation is normally bound to that of the respective Process Manager and, in fact, there exist quite a variety of different implementations. The *MPICH project* [17] for instance, which develops and maintains one of the two major open-source MPI implementations, featured a couple of different internal Process Managers over the years. Examples are *Gforker*, *MPD* [18] and *Hydra* [19], while the latter is still the most recent one. Furthermore, *Open MPI* [20], which is the other major open-source implementation, features its own Process Manager in terms of the *Open Run-Time Environment* (ORTE) [21]. In addition, non-vanilla and specialized MPI libraries frequently provide customized Process Managers that are tailored to their respective domains.

However, with PMI-1 [28], PMI-2 [29] and PMIx [30] for the *Process Manager Interface* there are at least standardization efforts for harmonizing the MPI session startup across different

MPI implementations. By utilizing one of those interfaces, it should in principle become possible to launch MPI executables with tools like `mpiexec` independently from the MPI library the executables are linked against.

ParaStation MPI and Process Manager

Although Hydra and ORTE are quite popular Process Managers for MPI runtime environments, we are focusing on ParaStation MPI and its Process Manager since it already features some experimental functionalities that are of particular interest for the goals of the project. Beside this, project partners ParTec and JUELICH are deeply involved in its development and therefore can more likely push development results from this project into the main releases.

Although *ParaStation MPI* (psmpi) is partly based on MPICH, it does not make use of the Hydra Process Manager but implements its own runtime environment. The runtime management facility of psmpi called ParaStation Process Management (psmgmt) offers a complete process management system that can in turn be combined with an external and more generic Resource Manager together with a batch queuing system plus Job Scheduler like Torque/Maui [31] or SLURM [32], the latter being the one selected for this project. In accordance to the roles specified above, the management tasks of psmgmt include creation of processes, distribution of startup information, monitoring the running sessions, and forwarding of I/O and signals across node boundaries.

The main part of this management framework is realised in a distributed fashion via the *ParaStation Daemons* (psid), that run in multiple instances on all nodes of the system as background processes. An important key to ParaStation's scalability is its inter-daemon communication subsystem that allows for an efficient message exchange between the psid instances via a highly-scalable Reliable Datagram Protocol (RDP). By this means the distributed daemon processes form a comprehensive management network that spans across the whole system. It is used for both: exchanging of inter-daemon commands and status messages as well as forwarding of I/O data and signals from and to the parallel MPI processes.

The Relation to the Resource Manager

As pointed out before, the Resource Manager is composed of the Process Manager in combination with an external Resource Allocator including a Job Scheduler that consistently partitions the cluster's resources into sub-domains according to the requested allocations of the submitted jobs. However, when calling ParaStation's `mpiexec`, the command initially contacts the local psid for determining the nodes and cores the requested MPI session should run on. Therefore, the additional information about predefined sub-partitions has here to be taken into account. For this purpose, psmgmt is able to limit the number of attainable nodes individually to the required subset. According to this, the Job Scheduler determines the set of resources assigned to an allocation, and psmgmt then translates this information into such a restricted view. New sessions that are launched within the allocation are then only spawned onto resources of this respective subset. However, for doing so, there needs to be some interface for the required information exchange between the Process Manager and the Job Scheduler. Following a *one daemon per cluster node* concept, the distributed daemons of ParaStation im-

plements such type of interface in terms of plugins that efficiently replace the native daemons of the higher-level Resource Managers on the nodes. For this a Torque-related plugin (psmom) and a SLURM-related plugin (psslurm) for the psids are provided. Both directly communicate with their respective Resource Managers and thus substitute the otherwise customary compute node daemons.

Figure 7 illustrates the orchestration between psmgmt and SLURM, as it is currently employed on the JURECA [34] system at the Jülich Supercomputing Centre in Germany. As one can see, the psid together with its psslurm plugin plays the central role regarding process startup and job control on the compute nodes: The SLURM Control Daemon (slurmctld) is running as usual on the master node, whereas on the compute nodes the psids have now adopted the part of the SLURM compute node daemons (slurmd). SLURM itself is designed to operate even in heterogeneous clusters with up to tens of millions of processors and can accept thousands of job submissions per second with a sustained throughput rate of hundreds of thousand jobs per hour. Its direct linkage to the network of distributed psids makes this orchestration between SLURM and ParaStation indeed highly scalable and very flexible.

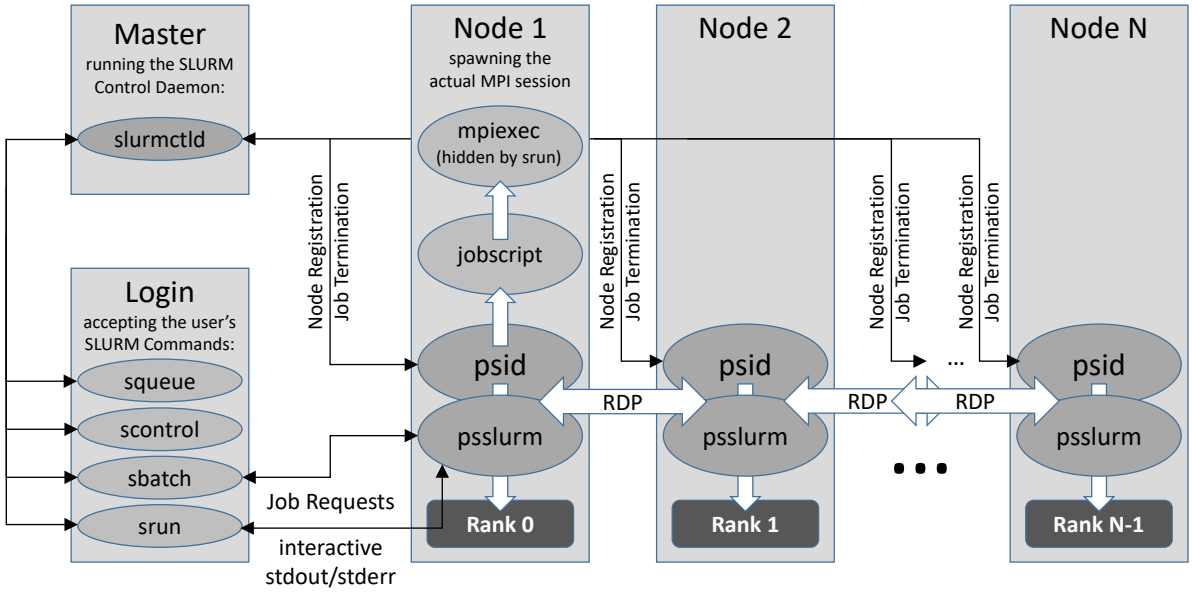


Figure 7: Orchestration of SLURM and ParaStation Management via its psslurm plugin

4.2 Requirements for System Software

After analysing the outcome of the application questionnaire as it has been presented in detail in D1.1, we could identify some requirements the DEEP-EST prototype applications will demand on the Resource Management System. These requirements can basically be divided into two orthogonal categories: On the one hand, there is the demand for handling of heterogeneous resources like accelerators, Network Attached Memory (NAM), network bridge nodes, and so forth — and, on the other hand, there is the requirement to support workflows for applications with multiple processing stages that are executed subsequently and/or concurrently within the

system modules.

Handling of Heterogeneous Resources

Since most applications will go more and more into using different types of accelerators for different tasks inside their workflow and thus become more and more heterogeneous, the Resource Management System needs to be able to manage these kinds of resources (including hardware accelerators, memory class and capacity, and storage system) and provide them to the jobs. Especially, this means that it has to handle different kinds of nodes with different kinds of accelerators, memory and storage systems connected to them. Furthermore, it has to be able to provide such different kinds of nodes to single job steps as requested by the user. This implies preparation of the nodes (e.g. optionally setup an instance of the BeeGFS on Demand (BeeOND) filesystem [22, 23] or possibly the configuration of an FPGA), management of processes running there and cleanup at the end. If the accelerator nodes are connected via special *gateways* or similar constructs (see Section 3), the Resource Management System has to manage these gateways as kind of global resource, too. This holds generally for all resources that are not represented by regular MPI processes. In summary it has to provide a flexible *horizontal* diversity of resources inside one job step (see Figure 8).

In case of an application failure that is using a multilevel checkpointing library [24, 25], it is important to cleanup the nodes, but without killing the user job as a whole, so the multilevel checkpointing library can restart the application from the local storage of these nodes, optionally using replacement nodes that have been pre-allocated or that can be allocated dynamically.

Besides that, the applications want to use different sets of resources in different steps of their workflows, in SLURM canonically executed as job steps. So ideally we need to be able to change the resource allocations not only between jobs, but in some way even between job steps, providing a *vertical* diversity of resources (see Figure 8).

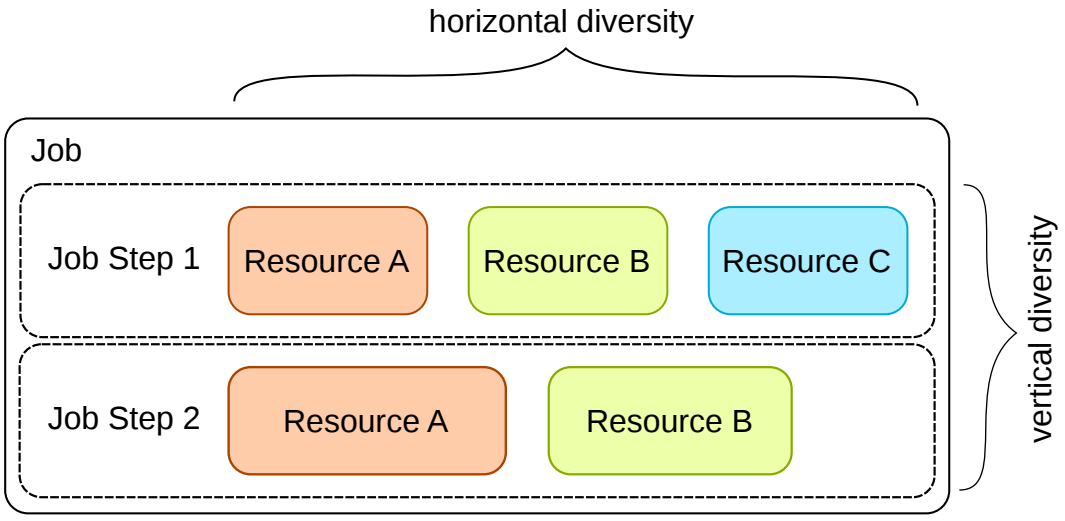


Figure 8: Horizontal and vertical resource diversity

A general topic pointed to by the application programmers is the requirement to have the re-

sources as stable as possible during their runtime. Even if this can only partly be guaranteed from the software side, it should be kept in mind to hold the number of potential error sources as small as possible — all the more so as stability is prone to diminish when heterogeneity increases.

Support for Application Workflows

Application workflows, as they are envisaged in this project, pose a challenge for both the Job Scheduler as well as the Resource Manager. While the former is responsible for generating appropriate schedules and resource allocations, the latter is especially in charge for the handling of the different resources assigned to the allocations and the management of the respective MPI sessions running within. According to this, in particular the Process Manager has to undertake the crucial task of handling sets of distributed processes as units by establishing sessions among them and keeping control over their lifetime as a group.

In doing so, this group and session management can actually be quite dynamic. For instance there might very well be multiple sessions running in parallel at the same time within a single allocation — probably demanding for *allocation-internal* scheduling policies then to be fulfilled by the Process Manager (and not by the Job Scheduler). Moreover, formerly distinct sessions may even join by calling `MPI_Comm_connect/accept`, as it has already been mentioned also in Section 3.2, and thus becoming a new and larger unit. Obviously, the Process Manager has to take such re-grouping operations into account in many regards: As the connection broker, a distributed but globally accessible *Naming Service* needs to be provided in order to allow formerly disconnected processes of separate sessions to join each other. Furthermore, as the process controller, bookkeeping of the grouping information must be provided in order to ensure an affiliation-oriented resource release and process clean-up.

In fact, with respect to system modularity, these aspects become even more challenging as workflow-related co-allocations are likely to range across module borders and to demand for a likewise comprehensive orchestration between the Process Manager and the Job Scheduler. E.g. in workflows, where every subsequent application step is being executed as a new session, the required overlap in co-scheduling for the data-passing from the previous step towards the new one needs to be coordinated between application, scheduler and runtime. In the same manner, if some of the applications requires the use of the `MPI_Comm_spawn` or the OmpSs offloading features [26, 27], the Resource Manager shall support the dynamic allocation of nodes in coordination with the Job Scheduler.

4.3 Draft for Design and Specification

Since the Modular Supercomputing Architecture envisioned in DEEP-EST will be heterogeneous, the Resource Management has to be able to start heterogeneous jobs, meaning jobs with different kind of executables on different kind of nodes with different kinds of resources like accelerators attached. The SLURM concept to handle that is called *Job Packs* and is coming just now with the newest version. We need to support and make use of this feature in every component of our Job Scheduling and Resource Management stack, adapting our de-

velopments to changes introduced by this new concept in SLURM, while maintaining dynamic process management and the related support for application workflows.

The related adaptations will in particular concern the psslurm plugin of the ParaStation Process Management (psmgmt), but might also affect further interfaces (confer also Figure 9). This also includes the interface for the interaction between the application processes and the management daemons. It may need to be widened in order to allow for the required orchestration between application, scheduler and runtime — as it has also been pointed out in the previous section. In fact, currently this interface is implemented for psmgmt according to the PMI-1 quasi-standard [28].

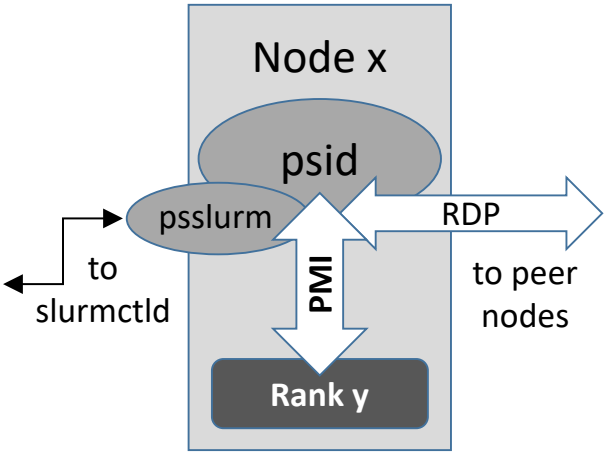


Figure 9: Interfaces of the psmgmt daemons

The PMI-1 API allows for the querying of group affiliation and group size as well as for the exchange of contact information via a common *Key/Value Space* (KVS). In addition, the API provides a means for handling the dynamic spawning of new process groups. However, PMI-1 does *not* explicitly provide an interface for managing or merely contacting a global Naming Service, as it would be required to support the joining of formerly independent process groups belonging to distributed sub-jobs. In contrast to this, the newer PMI-2 interface [29] also features functions for publishing contact information via a naming service that directly map onto the related MPI functions (e.g. `PMI2_Nameserv_publish/lookup` for `MPI_Publish/Lookup_name`) as well as functions for notifying the Process Manager about mergers and divorces of process groups (i.e. `PMI2_Job_connect` for `MPI_Comm_connect/accept` and `PMI2_Job_disconnect` respectively for `MPI_Comm_disconnect`). As it is quite likely that the DEEP-EST programming environment will at least partly rely on this ensemble of interface functions and runtime functionality, the envisioned Process Manager will most probably have likewise to implement this orchestration as a design specification — presumably by means of the PMI *Exascale* (PMIx) specification [30] that provides full PMI-1 and PMI-2 compatibility.

Another topic we need to deal with is the integration of NAM (network attached memory) in the Cluster. The NAM will be managed by a NAM Manager which provides information of the NAM usage to the scheduler and manages reservations inside of the NAM([15]). From the Resource Management's perspective the NAM will be a globally shared resource and it has to be able to allocate defined parts of this resource and to hand it over to the processes of a job. The reservations of parts of the NAM is the equivalent to malloc for the main memory. It might be necessary that the Process Manager will use libnam (the NAM software layer library) to call such a NAM malloc and then hand over the memory handle to the started processes. The details of this mechanism are to be discussed in the course of this project. Results shall be presented in the upcoming deliverable D5.2.

5 Job Scheduler

This chapter describes job scheduler functionality and the interfaces needed by application codes, benchmarks, and modelling tools.

5.1 Context and State of the Art

As it has been stated in previous chapters, the DEEP-EST project will use the widely used SLURM workload manager [32]. The main components of SLURM and the interaction with the resource manager have been described in Chapter 4. In this chapter we will focus on SLURM from the point of view of the Job Scheduler, the component responsible for job selection. SLURM includes a powerful plugin mechanism where most of the main SLURM functionalities can be modified, including a SLURM scheduler plugin [47]. Current versions include two basic options:

- builtin: This option is the default and it is a simple FIFO scheduling, where jobs are selected according to the arrival order. SLURM design includes one thread applying this basic policy and executed each time a new job arrives to the system, or a job finishes.
- backfilling: Backfilling is a scheduling policy widely used where jobs waiting in the queue can be advanced if they don't delay previously submitted jobs. Backfilling is in fact a strategy since it allows configuring it resulting in a more or less aggressive version of the same idea. For instance, we can configure the number of jobs considered to not be delayed, or the periodicity of the thread that applies backfilling. Jobs are selected for execution based on their resource requirements and these policy arguments [48]. Anyway it is clear that a key point regarding the job scheduler is the cooperation with the resource manager.

SLURM users typically request the desired resources through the user commands `sbatch`, `srun` and `salloc`. These commands include many options that must be analysed in order to see if they satisfy the requirements for a modular architecture.

There are four key concepts in the context of modular architectures that need to be supported by the job scheduler. These concepts (or requirements) have also been identified in previous chapters.

- Jobs are using resources in different modules
- Jobs are using non-computational resources such as Network Attached Memory (NAM)
- Jobs are adding/releasing resources dynamically, avoiding the necessity of wasting resources with advanced reservations. It might make sense to reserve resources in advance only in such cases when having these resources at specific moments is crucial
- Workflows: Jobs with dependencies between them.

The main DEEP-EST challenge is to offer an efficient and scalable module-aware (MA) scheduler providing a complete solution for users of Modular Supercomputer Architectures (MSAs) including all mentioned concepts. Furthermore, the MA-scheduler will support other actual and

important topics in current architectures such as being energy efficient, offering application checkpointing, etc. The DEEP-EST proposal will include both an MA-Scheduler architecture and policies in cooperation with the resource management solutions as proposed in Task 5.4.

Job scheduling support for modular or heterogeneous clusters has been mostly seen in cloud and grid contexts, but less or even not at all in the context of production HPC systems. In HPC systems heterogeneity is supported mainly by allowing the request of specific computational nodes, for instance, GPU nodes along with classical compute nodes.

5.1.1 Computational resources

Concerning more traditional computational resources, we have analysed the different SLURM options for resource specification and selection to check whether the already existing concepts can be eventually re-used or extended for use within MSAs context.

- **Constraint** allows requesting of nodes with specific features. Features are assigned to nodes previously by a system administrator. Resource counts as well as multiple constraints can be defined using logical AND or OR operators.
- **Generic consumable resources** These resources are configurable by the administrator. The user specifies the name and the count of the resources.
- **Reservations** allows the definition of node sets that can be allocated to users.
- **Partition** allows the specification of queues and thus part of the system related to that queue in which job allocation will be performed.
- **Heterogeneous jobs** (also called *job packs* in SLURM [49]) consist of several job components where each component can request specific resources. Initial support for heterogeneous jobs has been recently added to SLURM[1]. *Job packs* is a very interesting concept that must be deeply analysed to be used in our modular architecture for those jobs requiring resources in more than one module, either from the beginning or dynamically requested.

5.1.2 Non computational resources

SLURM also includes the possibility of requesting non-computational resources such as licenses for software, and network, switches, and memory requests. Network, switches, and memory requests are not computational resources but are hardware elements associated with computational nodes. However, we present them here since they could be extended to support different system components.

- Licenses. SLURM includes non-computational consumable resources such as licences. Licenses are used to control the number of cluster-wide software licences at the current version, but a similar concept could be used for hardware resources if needed.
- Network allows for specifying which type of the network should be used; options are system and blade.

[1] in version 17.11

- In a tree topology, option *switches* allows specifying the maximum count of network switches requested for job allocation.

5.1.3 Dynamic resource management

Regarding dynamically adding/releasing resources, SLURM offers the srun option *jobid* which allows for allocating a job step within an already running job. The srun option *dependency* allows for expanding of resources of an already running job. These mechanisms are not easy to use by SLURM users, but since the options already exist, that means that the functionality is internally considered in the SLURM code and that we may extend it in the project. Moreover, this feature can be leveraged to dynamically allocate replacement nodes in case of failure, so we can recover from a partial application failure using checkpoint/restart techniques without requiring pre-allocated sparse-nodes.

5.1.4 Workflows

Regarding workflows, i.e. set of jobs with dependencies, SLURM offers several mechanisms to specify dependencies: The *dependency* srun option allows for postponing the start of a job until some condition is met. For example, specific job(s) are either started or terminated successfully or not. Also, it allows for expanding of resources of already running jobs. The *multi-prog* srun option allows for running a job with different programs and each with different arguments.

5.2 Requirements for System Software

Based on D1.1 the targeted workload for MSAs will consist of two types of application scenarios:

- *Independent application* which is an application submitted to the system neither requesting input from other applications nor producing input to other applications.
- *Workflow* which is a set of applications executed with mutual dependencies, i.e. some applications from the set may need input from others. It might be expected that the total workflow set is not known in advance. Potentially, applications in the set may have some “real-time” requirements. An example of workflow application from WP1 is ASTRON.

Regarding the timeline in which resources are requested by a single application as well as the types of resources the application requests we identified the following possibilities:

- Applications may ask for resources in a specific module at submission time. For example, NEST Benchmark will require CM nodes.
- Applications may ask for resources in more than one module at submission time. Before the applications are ready for dynamic resource requests they may ask for more than one module at submission time.
- Applications may ask for additional resources in the same and/or a different module during the execution. For example, in the case of GROMACS there is a plan to request CM nodes at submission time and later on request ESB and DAM nodes for the offload computation.

- Applications may ask not only for computing resources. They may ask for memory, constraints, an instance of the BeeOND filesystem, licences and specific resources already supported by SLURM. For example, the SpaceWeather application from KU Leuven may request NAMs. The applications from the University of Island may require NAMs and GCEs.

5.3 Draft for Design and Specification

There are two main decisions to be made at this moment regarding the changes we need to make in SLURM to support MAs, and these are (1) architectural changes, and (2) functionality changes.

Introducing changes in SLURM's architecture may pose the problem of incorporating DEEP-EST contributions in the SLURM distribution. For that reason, we have evaluated the cost of the scheduling policy in SLURM as a function of the system load to estimate the potential SLURM limits when running in a modular architecture.

We performed a set of experiments using the SLURM simulator developed initially at BSC [35], and passed several iterations of improvements, at CSCS [36], and at Umea University/Berkley Lab [37]. Recently, in the context of DEEP-EST at BSC, we have introduced additional improvements to speed up the simulator itself and to validate it comparing to results reported by SLURM when doing real workload executions.

We have generated a workload trace file using the model proposed by Cirne and Berman in 2001 [38] based on the analysis of many workloads coming from real workload traces. This model includes user behaviour concerning submission patterns, job sizes according to system size, system load, etc. We have generated a workload with 10k jobs in a system with 10k nodes with 8 CPUs per node. To evaluate the scheduling cost, we have simulated the workload trace with the default system size of 10k nodes, and, also, changing the system size to increase or reduce the system load synthetically.

The SLURM scheduler comprises two types of scheduling strategies: (1) the **normal** scheduler, executed each time a new job arrives to the system, and each time a job finishes its execution, and (2) the **backfilling** scheduler, executed periodically to decide if some of the queued jobs can be started without penalising the start time of previously arrived jobs. The backfilling scheduler can be configured by setting the period after which the scheduler is executed, and the number of jobs in the queue considered to be backfilled. In these experiments the worst case has been evaluated where all the jobs in the queue are considered. Of course, this is not realistic, since as we reduce the system size, thereby increasing the number of jobs being queued up, the cost of backfilling will increase. To perform our evaluation, we measured, per scheduler, both normal and backfill, the following metrics:

- Average time per entry of the scheduler. The average time, in *ms*, consumed by scheduler each time the scheduling function is executed.
- Average slowdown. The slowdown is calculated as job's response time, i.e. time between the job's submission and job's end, divided by job's actual duration. It reflects the delay suffered by jobs when executed in a real system. The metric gives us a hint about the

load of the system.

- Total number of backfilled jobs, and average size of the queue. These metrics will give us an idea of the amount of work for the backfilling scheduler.

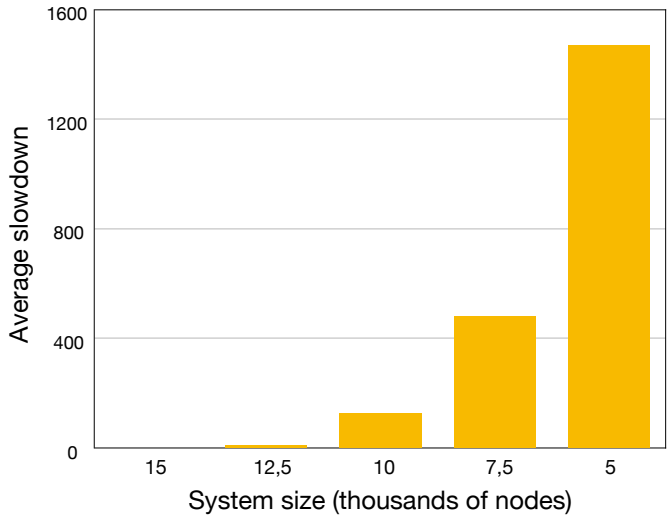


Figure 10: Average slowdown

Figure 10 shows the average slowdown for the different system sizes. The system of 10k nodes is the reference. We can observe how increasing the system size and running the same workload results in a reduction in the average slowdown, since the average wait time is reduced, i.e. with the system size of 15k nodes the average wait time is zero, resulting in the average slowdown being equal to one. On the other side, reducing the system size has a significant negative impact on performance. Reducing the system size by a factor of two, i.e. from 10k to 5k nodes, increases the average slowdown from 127 to 1471.

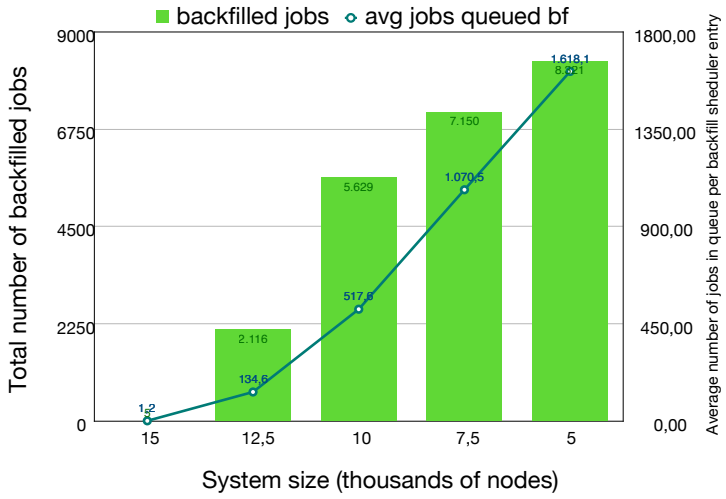


Figure 11: Total number of backfilled jobs and queue size

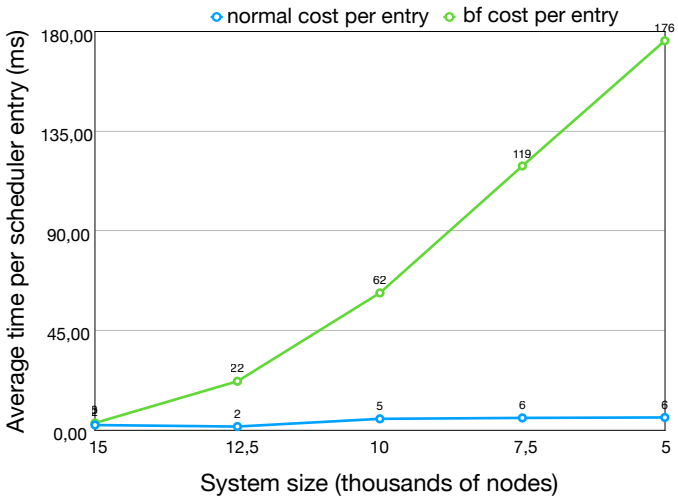


Figure 12: Execution time (ms), each time normal and backfilling scheduler are executed

Figure 11 shows in a single graph the number of backfilled jobs and the average queue size for the different system sizes. The number of backfilled jobs is reduced when the system is increased, since jobs can be started by the normal scheduler. However, with a smaller system, the system is highly loaded and, since jobs have to wait a lot of time in the ready queue, the backfilling scheduler has more opportunities to start short jobs while big jobs are still running. In the worst case, the average number of jobs in the ready queue is 1618 jobs. Since the backfilling algorithm is configured to consider all queued jobs, results presented in Figure 12 are as expected. We can see how the cost of the normal scheduler is quite constant and not significantly increased, neither with the load nor the system size. On the other side, backfilling costs depends on the number of jobs in the queue (in the simulated configuration). Given the number of jobs queued, we can estimate that the cost of the backfill scheduler is 0.1 *ms* per queued job.

Given these results, we can conclude that the scheduling cost is not significantly increased with system size. Moreover, we can think of a system with one core or even one node dedicated to execute the scheduler. We propose to start with the current architecture, considering the different modules as different types of computational nodes or consumable “resources”.

The next steps will consist of refining the use cases to be executed with the already existing SLURM concepts to detect potentially missing functionalities to be included in SLURM.

6 System Monitoring and RAS Plane

One of the major design criteria of the Modular Supercomputing Architecture (MSA) revolves around energy efficiency. Similarly to the first two iterations of the project (DEEP and DEEP-ER), the main objective behind DEEP-EST is to propose an alternative architectural concept of an HPC system (covering both hardware and software requirements) that could be feasible for the realization of a future Exascale platform; it is well established that this is not practicable without first appropriately addressing the significant challenge posed by the power consumption that an Exascale machine could potentially draw. One of the essential steps to tackle this issue is to gather in-depth knowledge of the inner workings of such a machine and to understand how much power is spent when, where, and on what and what could be done to minimize it. To get this insight, a powerful system monitoring facility is required that acquires as much sensor data on a system as possible and as exhaustive as feasible.

There are several additional perks for developing such a tool. For example, information of power and/or energy consumption could be fed to the resource and job management system with the opportunity of developing novel energy-aware scheduling strategies (e.g., power capping policies, dynamic frequency scaling for energy-efficient job runs, etc.). Additionally, exposing the energy consumed by jobs could contribute to an energy-efficient operation of the system, e.g, by defining accounting policies that can stimulate users to optimize their code with the objective of minimizing the energy consumed by their applications. Last but not least, correlation analysis between different system properties and parameters (such as power consumption, CPU frequency, temperature, etc.) could provide insights on the behaviour of future HPC systems potentially based on the same modular architecture.

Being a “first-of-a-kind” experimental platform thanks to its MSA design, the DEEP-EST prototype will be an excellent candidate testbed for experimenting with novel features in our monitoring solution. Specifically, as a follow-up to the work already done in DEEP, the main objective for the “System Monitoring and RAS Plane” task will centre on improving the Data Centre Data Base (DCDB) tool, the sensor data monitoring and storage tool developed by BAdW-LRZ. In this chapter we will briefly outline where we are, what we need and what will be improved in DCDB. Specifically, in Section 6.1 we will provide background information on the state-of-the-art of system monitoring and describe the current implementation of DCDB. Section 6.2 will list the requirements such a system monitoring solution needs to fulfil from an application developer’s point of view as well as from a system operator’s. Finally, in Section 6.3 we will provide a preliminary specification of the system software design for DCDB and present new desired features that need to be developed for enhancing the tool.

6.1 Context and State of the Art

Many system monitoring tools that are currently deployed on HPC systems are provided by system vendors and specific to their platform. They usually only cover the operational status of the supercomputer, such as the health of all hardware components and subsystems, the load, and in some cases even information on its power consumption. Often, HPC centres use multiple monitoring tools on different systems, hence they are hardly getting a comprehensive overview

on all of their operations. Furthermore, these monitoring solutions typically concentrate on IT components only, leaving building infrastructure information (such as cooling and power delivery) out of scope or being tracked with additional monitoring solutions. Though most tools store historical data, access to this data is typically not readily available or documented. This is particularly inconvenient as the analysis features of vendor-provided monitoring tools often are very limited and do not allow for more than plotting graphs of historical data.

Open-source monitoring solutions such as Ganglia [39] and Icinga [40] exist as well and could potentially be used to not only monitor the IT systems but also their infrastructure. However, these solutions are primarily focused on giving an overview on the current system status. Analysis of historical data in these tools is very limited and further impeded by using storage-constrained databases (such as RRDtool [41]) that average old sensor data over long periods (typically a couple of hours) to save space.

However, to perform an extended system analysis and to get an in-depth understanding of its operational characteristics, a useful system monitoring solution should be more sophisticated than the above mentioned existing ones:

i) it must provide an overview of the whole HPC centre, not only of (some) IT systems;

ii) sensor data must be available over long periods of time with high granularity;

iii) any component that provides sensor data should be monitored as its usefulness may only become clear after having it available;

iv) data must be easily accessible for analysis.

The DCDB monitoring tool [42] has been developed within the DEEP project with the above listed requirements in mind. Specifically, a key design criterion has been to collect as much sensor information as possible without discarding any data, but rather filtering it only during the analysis. DCDB consists of three major software components: one or multiple data sources (defined as *Pushers*) responsible for acquiring sensor data from the monitored system devices. Once retrieved, the sensor data is sent to one or more *Collect Agents*, which are responsible for gathering all monitored data from the Pushers and storing it persistently in a key-value store, from where it can be conveniently retrieved for post-processing operations and data analysis.

Additional details on the design and specification of the software architecture of DCDB are presented in Section 6.3.

6.2 Requirements for System Software

The initial co-design approach adopted in DEEP-EST resulted in the identification of a set of application requirements that system software developers will attempt to satisfy. The interested reader can find more details about them in Deliverable D1.1 “Application Co-Design Input”. Apart from some notable exceptions, application partners provided relatively modest feedback regarding system monitoring, with particular emphasis on power and energy consumption. In general, all the interested partners considered it very useful having knowledge of the power and energy consumed by their application runs; some partners even advanced the idea of increasing the time resolution of the power consumption measurement process, in order to better

analyse the behaviour of specific compute kernels of their application at relatively small time scales. Since this kind of resolution will be significantly dictated by hardware-specific decisions, the possibility of implementing such a solution will be investigated in joint collaboration with our system integrator and project partner Megware.

From a system administration perspective, monitored sensor data should also be easily available via command-line interfaces, user-space libraries, and possibly over network. For making use of the collected data, a set of convenient analysis tools will need to be provided as well, both for system operations and for application developers.

The information related to RAS provided by the system monitoring could be used by checkpointing libraries to determine parameters such as the optimal checkpoint interval. This could be dynamically depending on the latest statistics on failures. It has been demonstrated that there are periods of higher failure frequency in datacentres and supercomputers, and using the latest statistics provided by the RAS system could help checkpointing libraries detect these periods and increase the reliability of the runs (e.g. higher checkpoint frequency, stronger erasure codes, etc). There is also proof of difference in failure behaviour between nodes of the same machine: some nodes are more prone to failures due to manufacturing irregularities. This information, if provided, could be use to harden execution in those nodes.

6.3 Draft for Design and Specification

The software design for the *System Monitoring and RAS Plane* in DEEP-EST will follow the design inherited from the DEEP project as briefly outlined in Section 6.1 and depicted in Figure 13: the central components of the system monitoring are the Apache Cassandra key-value store [43] for storing sensor data persistently, the Pushers that acquire sensor data from different sources or devices, and the Collect Agents that collect the sensor information from the Pushers and forward them to Cassandra.

Cassandra provides an open-source, scalable, distributed data store with high availability and no single point of failure and can be considered a hybrid between a NoSQL key-value store and a tabular database. Cassandra is mainly conceived as a distributed solution featuring the opportunity of deploying large numbers of nodes across several commodity servers.

In general, DCDB implements a push paradigm for retrieving monitoring data: instead of a single central daemon that collects data from different sources and stores it to a database, a set of distributed Pusher daemons close to the actual sensors push the data to the Collect Agent. Currently, the following Pusher daemons are available:

- an IPMI Pusher, for retrieving possible standard “out-of-band” IPMI metrics, like power consumption, temperatures, fan speeds, etc.
- an SNMP Pusher, for acquiring standard “out-of-band” SNMP metrics, such as power drawn by network switches or PDUs
- a sysfs Pusher, for “in-band”-available system information, like CPU frequencies and temperatures or specific data provided by dedicated meters from system integrators (e.g., energy meters)

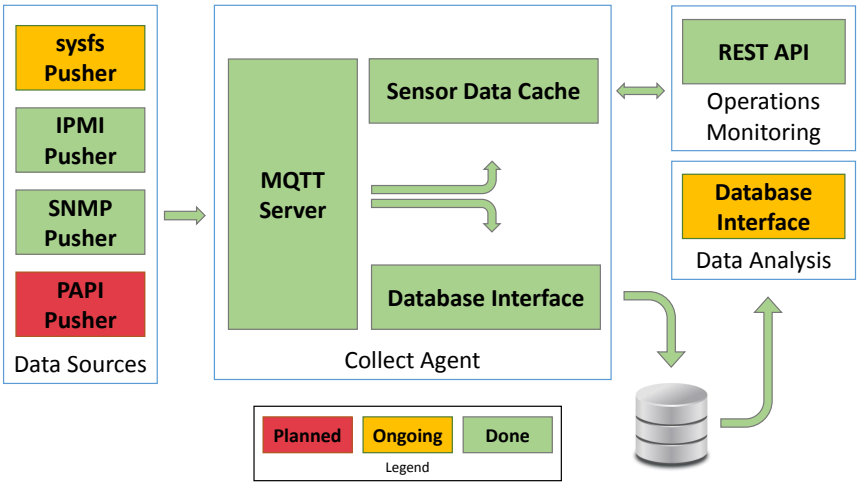


Figure 13: Current status of the software architecture of DCDB

Although some of those Pushers implement a pull scheme themselves to actually gather the monitoring data via established protocols such as IPMI or SNMP, others (such as the sysfs Pusher) with access to in-band data strictly follow the push paradigm.

Communication between a Pusher and a Collect Agent is based on MQTT, the Message Queue Telemetry Transport protocol. MQTT is an ISO standard and an established protocol for telemetry data and used in a wide range of IoT applications. It employs a publish/subscribe model, where data sources publish their data on a broker and interested parties can subscribe to topics of interest. For the purposes of DCDB, the pusher daemons publish their monitoring data under well-defined topics that precisely describe the source of the monitoring data. The Collect Agent receives all MQTT messages from its associated Pushers, translates their topics into unique database keys, and stores the actual sensor data within the message in the Cassandra database. As the Collect Agent would be the single subscriber to all topics it omits the actual role of subscriber but instead assumes the role of a broker and consumes all received MQTT messages itself which improves performance and reduces code complexity.

DCDB is designed to be fully distributed to allow for load balancing and improved reliability: Cassandra by design is a distributed key-value store that can be spread across multiple database servers with optional data redundancy. Multiple instances of the Collect Agent as well as the individual Pushers can be distributed across a network. Any Collect Agent may communicate with any database server and any Pusher with any Collect Agent. Cassandra is configured such that the MQTT topic (that translates to a 128bit unique key) of any given sensor determines the database server that is responsible for actually storing the data. Consequently, there is a natural, performance-optimal relation of Pushers, Collect Agents and Cassandra servers which will be configured by default. However, for load balancing or fail-over purposes,

that may be changed at any given time.

This core functionality of DCDB described above has mostly been implemented during and (to a limited extend) after the DEEP project and can be considered a proof-of-concept implementation. In the DEEP-EST project, the development will concentrate mainly on bringing DCDB closer to a production-ready software solution and develop it further in terms of performance and productivity. New Pusher daemons will be provided where necessary to make all sensors implemented in the DEEP-EST prototypes available in DCDB. Also, convenient access to the "energy-to-solution" of completed application runs will be fulfilled by implementing a suitable SLURM plugin.

To further facilitate system administrators for performing standard monitoring of the system, a convenient REST API along with a web user interface will be implemented and deployed on the DEEP-EST prototype. The web UI will be beneficial also for application developers that are interested in tracking the progress of their jobs, potentially during runtime. In this regard, further support will be provided by integrating the user interface with sophisticated tools for analysing applications behaviour and detecting potential correlations between different code phases and system metrics (e.g., energy consumption, temperature, etc.). We are currently in the process of evaluating existing visualisation frameworks such as Grafana [44], Kibana [45], and Timelion [46] to see whether they can be leveraged for this purpose in DCDB.

7 Summary

This document gives a first overview on the work planned within WP5. For this, results extracted from a questionnaire addressing the application developers of WP1 were collected in order to identify requirements on the system software of a Modular Supercomputer Architecture. The resulting system software to be developed in the course of the DEEP-EST project will be crucial for the operation of the prototype hardware to be developed in WP3 and WP4.

The identified requirements have to be seen as preliminary. Main reason for this is the fact that at least part of the system software strongly depends on the actual hardware chosen for the realisation of the MSA. Even though first ideas on the actual hardware architectures are sketched in D3.1 written at the same time as this document, final decisions on the prototype cannot be expected before project month 12 when they will be summarised in the final hardware architecture document D3.2. Nevertheless, the requirements that we were able to identify at this early stage will provide a first idea on the system software architecture to be realised for the MSA prototype, even though details still might be subject of change.

The document discusses draft design decisions for the various components along the line of the tasks foreseen for WP5 in the project's description of work. The most important decisions include:

- For the Interconnect Management most efforts will concentrate on the support of the novel fabri^3 hardware to be implemented by EXTOLL. Since fabri^3 will provide an integrated management CPU capable to directly access the Tourmalet chips via an I^{2}C bus EMP has to be adapted. Furthermore, the integration of the NAM and GCE hardware into the fabri^3 will require additional efforts on system software for optimal support.
- Design decisions for the Inter-Module Network Bridging are particularly hard to draw due to the fact that those will strongly depend on the overall network topology of the DEEP-EST prototype, especially on the choice of networking technologies. Those decisions still have to be made. Therefore, this document presents more general thoughts on possible fabric choices and implications on the software design. This can be seen as part of the co-design efforts between WP5 and WP3.
- Concerning the Resource Management design decision are well advanced due to the fact the they will not depend on the details of the hardware implementation. The high-level architecture will be based on the combination of SLURM and ParaStation. Since support for heterogeneous architectures like the MSA was added to SLURM just recently, the detailed planning of according changes to both, SLURM and ParaStation are still due.
- On the side of the Job Scheduler the fundamental decision was drawn to stick with SLURM here. By leveraging a simulation tool developed by partner BSC we were able to exclude possible performance bottlenecks of the corresponding part of the SLURM software. Therefore, this task will concentrate of the implementation of the different feature requests extracted from the questionnaire. These include a more intelligent support for workflows, more capabilities directed to dynamic resource requests, support for non-compute resources like NAM capacities, etc.
- For the task on System Monitoring and RAS Plane efforts will concentrate on stabilisation

and consolidation of software products developed in the course of the DEEP series of projects. This will also include support for the specific hardware to be developed in the course of the project.

A detailed design of the system software architecture will be presented in WP5's next deliverable D5.2.

List of Acronyms and Abbreviations

A

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit, Integrated circuit customised for a particular use
ASTRON	Netherlands Institute for Radio Astronomy, Netherlands

B

BADW-LRZ	Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften. Computing Centre, Garching, Germany
BDA	Big Data Analytics
BDEC	Big Data and Extreme-Scale Computing
BeeGFS	The Fraunhofer Parallel Cluster File System (previously acronym FhGFS). A high-performance parallel file system
BeeOND	BeeGFS-on-demand, parallel storage based on BeeGFS
BIC	Booster Interface Card (gateway nodes in DEEP)
BN	Booster Node (functional entity)
BoP	Board of Partners for the DEEP EST project
BSC	Barcelona Supercomputing Centre, Spain
BSCW	Repository used in the DEEP EST project to share all project documentation

C

CA	Consortium Agreement
Cassandra	The Apache Cassandra key-value store
CERN	European Organisation for Nuclear Research / Organisation Européenne pour la Recherche Nucléaire, International organisation
CLI	Command-Line Interface (a terminal/console-based user interface)
CM	Cluster Module: with its Cluster Nodes (CN) containing high-end general-purpose processors and a relatively large amount of memory per core
CME	Coronal Mass Ejections
CMS	Compact Muon Solenoid experiment at CERN's LHC
CN	Cluster Node (functional entity)
CNN	Convolutional Neural Networks
COTS	Commercial off-the-shelf
CPU	Central Processing Unit

CSIC Spanish Council for Scientific Research

D

DAM Data Analytics Module: with nodes (DN) based on general-purpose processors, a huge amount of (non-volatile) memory per core, and support for the specific requirements of data-intensive applications

DCDB Data Centre Data Base (a tool developed in DEEP)

DDG Design and Developer Group of the DEEP-EST project

DEEP Dynamical Exascale Entry Platform (project FP7-ICT-287530)

DEEP-ER DEEP - Extended Reach (project FP7-ICT-610476)

DEEP/-ER Term used to refer jointly to the DEEP and DEEP-ER projects

DEEP-EST DEEP - Extreme Scale Technologies

Dimemas Performance analysis tool developed by BSC

DN Nodes of the DAM

DNN Deep neural network

DoW Description of Work

DSL Domain-specific Language

DRAM Dynamic Random Access Memory. Typically describes any form of high capacity volatile memory attached to a CPU

E

EC European Commission

EEHPC Energy Efficient High Performance Computing

EEP European Exascale Projects

EMP EXTOLL Management Process

EPT4HPC European Technology Platform for High Performance Computing

ESB Extreme Scale Booster: with highly energy-efficient many-core processors as Booster Nodes (BN), but a reduced amount of memory per core at high bandwidth

EU European Union

Exascale Computer systems or Applications, which are able to run with a performance above 10^{18} Floating point operations per second

EXDCI European Extreme Data & Computing Initiative

EXN The EXTOLL Linux Ethernet emulation layer

EXTOLL High speed interconnect technology for HPC developed by UHEI

Extrae Performance analysis tool developed by BSC

F

fabri^3	Interconnect technology based on EXTOLL (pron. “Fabri-Cube”)
FFT	Fast Fourier Transform
FHG-ITWM	Fraunhofer Gesellschaft zur Foerderung der Angewandten Forschungs e.V., Germany
Flop/s	Floating point Operation per second
FP7	European Commission 7th Framework Programme
FPGA	Field-Programmable Gate Array, Integrated circuit to be configured by the customer or designer after manufacturing
FTI	Fault Tolerant Interface, a checkpoint/restart library

G

GCE	Global Collective Engine, a computing device for collective operations
GFlop/s	Gigaflop, 10^9 Floating point operations per second
GLA	General Learning Algorithms
GPU	Graphics Processing Unit
GROMACS	A toolbox for molecular dynamics calculations providing a rich set of calculation types, preparation and analysis tools

H

H2020	Horizon 2020
HBM	High Bandwidth Memory
HPC	High Performance Computing
HPDA	High Performance Data Analytics
HPDBSCAN	A clustering code used by UoI in the field of Earth Science
HW	Hardware
Hydra	The MPICH-native Process Manager

I

IC	Innovative Council
I^{2}C	Inter-Integrated Circuit computer bus
IB	see InfiniBand
IDC	International Data Corporation
InfiniBand	A networking communication standard for HPC clusters
Intel	Intel Germany GmbH, Feldkirchen, Germany
I/O	Input/Output. May describe the respective logical function of a computer system or a certain physical instantiation
IP	Intellectual Property
IPMI	Intelligent Platform Management Interface

iPic3D	Programming code developed by the KULeuven to simulate space weather
ISO	International Organisation for Standardisation

J

JLESC	Joint Laboratory for Extreme Scale Computing
JUBE	Jülich Benchmarking Environment
JUELICH	Forschungszentrum Jülich GmbH, Jülich, Germany
JURECA	Jülich Research on Exascale Cluster Architectures

K

KNL	Knights Landing, second generation of Intel® Xeon Phi (TM)
KNH	Knights Hill, next generation of Intel® Xeon Phi (TM)
KULeuven	Katholieke Universiteit Leuven, Belgium

L

LHC	Large Hadron Collider (LHC), the world's most powerful accelerator providing research facilities for High Energy Physics researchers across the globe
libNAM	Software layer for accessing and managing NAM (Network Attached Memory) modules
LLNL	Lawrence Livermore National Laboratory
LOFAR	Low-Frequency Array, an instrument for performing radio astronomy built by ASTRON

M

Megware	Megware Computer Vertrieb und Service GmbH, Chemnitz, Germany
MHD	Magneto-hydrodynamics
Mont-Blanc	European scalable and power efficient HPC platform based on low-power embedded technology
MoU	Memorandum of Understanding
MPI	Message Passing Interface, API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages
MPICH	MPI implementation maintained by Argonne National Laboratory
MSA	Modular Supercomputer Architecture

MUSIC	Multisimulation Coordinator (MPI-based library for coupled codes)
MQTT	Message Queuing Telemetry Transport (a publisher/subscriber-based messaging protocol)

N

NAM	Network Attached Memory
NCSA	National Centre for Supercomputing Applications, Bulgaria
NEST	Widely-used, publically available simulation software for spiking neural network models developed by NMBU
NF	Network Federation within the DEEP EST prototype
NMBU	Norwegian University of Life Sciences, Norway
NN	Neural Network
NUMA	Non-Uniform Memory Access
NV-DIMM	Non-Volatile Dual In-line Memory Module
NVM	Non-Volatile Memory. Used to describe a physical technology or the use of such technology in a non-block-oriented way in a computer system
NVRAM	Non-Volatile Random-Access Memory

O

OA	Open Access
ODC	Other direct costs
OGC	Open Geospatial Consortium
OmpSs	BSC's Superscalar (Ss) for OpenMP
Omni-Path Architecture	Communication architecture owned by Intel
OPA	see Omni-Path Architecture
OpenCL	Open Computing Language, framework for writing programs that execute across heterogeneous platforms
openHPC	A community effort that is initiated from a desire to aggregate a number of common ingredients required to deploy and manage HPC Linux clusters
OpenMP	Open Multi-Processing, Application programming interface that support multi-platform shared memory multiprocessing
Open MPI	MPI implementation maintained by the Open MPI Project
ORTE	Open MPI Runtime Environment (i.e. a Process Manager)

P

ParaStation	Software for cluster management and control developed by JUELICH and its linked third party ParTec
Paraver	Performance analysis tool developed by BSC
ParTec	ParTec Cluster Competence Center GmbH, Munich, Germany. Linked third Party of JUELICH in DEEP EST
PCIe	Peripheral Component Interconnect Express (a high-speed serial computer expansion bus standard)
PDU	Power Distribution Unit
PFlop/s	Petaflop, 10^{15} Floating point operations per second
Phi	see Xeon Phi
PI	Principal Investigator
piSVM	Parallel classification algorithm
PME	Particle mesh Ewald
PMI	Process Management Interface
PMT	Project Management Team of the DEEP-EST project
PRACE	Partnership for Advanced Computing in Europe (EU project, European HPC infrastructure)

Q

R

R&D	Research and Development
RAM	Random-Access Memory
RAS	Reliability, Availability, Serviceability
RDA	Research Data Alliance
RDMA	Remote Direct Memory Access / Remote DMA-based Memory Access
RDP	Reliable Datagram Protocol
REST	Representational State Transfer (an interface for web services)
RM	Resource Manager
RMA	Remote Memory Access
RMI	Remote Method Invocation
RML	Risk management list used in the DEEP-EST project

S

SCR	Scalable Checkpoint/Restart. A library from LLNL
SDV	Software Development Vehicle: HW systems to develop software in the time frame where the DEEP-EEST prototype is not yet available

SIMD	Single Instruction Multiple Data
SIONlib	Parallel I/O library developed by Forschungszentrum Jülich
SKA	Square Kilometer Array
SLURM	Job scheduler that will be used and extended in the DEEP-EST prototype
Slurm	MHD code developed by KULeuven
SME	Small and Medium Enterprises
SNMP	Simple Network Management Protocol
SRA	Strategic Research Agenda prepared by ETP4HPC
SSSM	Scalable Storage Service Module
STEM	Science, technology, engineering and mathematics
STS	Satellite time series
SW	Software

T

TCP/IP	Transmission Control Protocol and the Internet Protocol (a protocol family)
TensorFlow	Open-source software library for dataflow programming
TFlops	Teraflop, 10^{12} Floating point operations per second
ThinkParQ	Spin-off company of FHG ITWM
Tk	Task, Followed by a number, term to designate a Task inside a Work Package of the DEEP-EST project
ToW	Team of Work Package leaders of the DEEP-EST project
TRL	Technology Readiness Levels

U

UEDIN	University of Edinburgh, UK
UHEI	Ruprecht-Karls-Universitaet Heidelberg, Germany
UI	User Interface
UoI	Háskóli Íslands University of Iceland, Iceland
UPC	Universitat Politècnica de Catalunya. Barcelona, Spain

V

W

WLCG	Worldwide LHC Computing Grid

WP Work package

X

x86 Family of instruction set architectures based on the Intel 8086 CPU
Xeon Non-consumer brand of the Intel® x86 microprocessors (TM)
Xeon Phi Brand name of the Intel® x86 manycore processors (TM)

Y

Z

Bibliography

[1] The Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard – Version 3.1*, June 2015

[2] Marc-Oliver Gewaltig and Markus Diesmann: *NEST (NEural Simulation Tool)*, Scholarpedia, issue 2(4), 2007, http://www.scholarpedia.org/article/nest_(neural_simulation_tool)

[3] Espen Hagen et al.: *Hybrid Scheme for Modeling Local Field Potentials from Point-Neuron Networks*, Cerebral Cortex Journal, issue 26(12), 2016, http://dx.doi.org/10.1093/cercor/bhw237

[4] *Arbor: The Arbor multi-compartment neural network simulation library* [Online], Available: https://github.com/eth-cscs/arbor

[5] *MUSIC: The Multisimulation Coordinator* [Online], Available: https://github.com/INCF/MUSIC

[6] *Elephant: Electrophysiology Analysis Toolkit* [Online], Available: http://elephant.readthedocs.io/en/latest

[7] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen: *GROMACS: A message-passing parallel molecular dynamics implementation*, in Computer Physics Communications, volume 91, 1995, https://doi.org/10.1016/0010-4655(95)00042-E

[8] M. Goetz, C. Bodenstein and M. Riedel: *HPDBSCAN – Highly Parallel DBSCAN* in Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'15), Machine Learning in HPC Environments (MLHPC) Workshop https://dl.acm.org/citation.cfm?id=2834894

[9] Abadi, Martín et al.: *Tensorflow: Large scale machine learning on heterogeneous distributed systems* arXiv preprint, arXiv:1603.04467, 2016

[10] B. Bierbaum, Clauss, M. Pöppe, S. Lankes, and T. Bemmerl: *The new Multidevice Architecture of MetaMPICH in the Context of other Approaches to Grid-enabled MPI*, in Proceedings of the 13th European PVM/MPI Users Group Meeting (EuroPVM/MPI2006), volume 4192 of LNCS, 2006, http://www.springerlink.com/content/871152j77t331234

[11] S. Pickartz, C. Clauss, S. Lankes, S. Krempel, T. Moschny, and Antonello Monti: *Non-Intrusive Migration of MPI Processes in OS-bypass Networks*, IEEE Parallel and Distributed Processing Symposium Workshops (IPDPSW), IPRDM Workshop, 2016, http://dx.doi.org/10.1109/IPDPSW.2016.134

[12] Hans-Christian Hoppe and Herbert Cornelius: *DEEP-EST WP3/4 Technology Evaluation Discussion*, November 2017

[13] Gilad Shainer: *Offloading vs. Onloading: The Case of CPU Utilization* [online], in HPCwire: Global News and Information on High Performance Computing, 18th June 2016, https://www.hpcwire.com/2016/06/18/offloading-vs-onloading-case-cpu-utilization/

[14] Matthew G. F. Dosanjh, Ryan E. Grant, Patrick G. Bridges, and Ron Brightwell: *Re-evaluating Network Onload vs. Offload for the Many-Core Era*, in Proceedings of the

IEEE International Conference on Cluster Computing (CLUSTER), September 2015, http://www.cs.unm.edu/~mdosanjh/6598a342.pdf

[15] J. Schmidt and Andreas Galonska: *DEEP-ER WP3: Network Attached Memory (NAM)*, 2016

[16] Dror Feitelson and Larry Rudolph: *Toward Convergence in Job Schedulers for Parallel Supercomputers*, in Job Scheduling Strategies for Parallel Processing: Proceedings of the IPPS'96 Workshop, Springer, Lecture Notes in Computer Science (LNCS), volume 1162, pages 1–26, April 1996, http://dx.doi.org/10.1007/BFb0022284

[17] William Gropp, Ewing L. Lusk, Nathan E. Doss, and Anthony Skjellum: *A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard*, in Parallel Computing, volume 22, number 6, pages 789–828, 1996, http://dx.doi.org/10.1016/0167-8191(96)00024-5

[18] Ralph Butler, William Gropp, and Ewing L. Lusk: *A Scalable Process-Management Environment for Parallel Programs*, in Recent Advances in Parallel Virtual Machine and Message Passing Interface: Proceedings of the 7th European PVM/MPI Users' Group Meeting, Springer, Lecture Notes in Computer Science (LNCS), volume 1908, pages 168–175, September 2000, http://dx.doi.org/10.1007/3-540-45255-9_25

[19] Argonne National Laboratory, Mathematics and Computer Science Division: *Hydra Process Management Framework: Implementation and Control Flow* [online], Wiki Page, https://wiki.mpich.org/mpich/index.php/Hydra_Process_Management_Framework

[20] E. Gabriel et al.: *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation*, in Recent Advances in Parallel Virtual Machine and Message Passing Interface: Proceedings of the 11th European PVM/MPI Users' Group Meeting, Springer, Lecture Notes in Computer Science (LNCS), volume 3241, pages 97–104, September 2004, http://dx.doi.org/10.1007/978-3-540-30218-6_19

[21] R. H. Castain, T. S. Woodall, D. J. Daniel, J. M. Squyres, B. Barrett, and G. E. Fagg: *The Open Run-Time Environment (OpenRTE): A Transparent Multicluster Environment for High-performance Computing*, in Future Generation Computing Systems, volume 24, number 2, February 2008, http://dx.doi.org/10.1016/j.future.2007.03.010

[22] Fraunhofer Center for High Performance Computing: *BeeGFS – The Parallel Cluster File System* [online] https://www.beegfs.io/content/

[23] Fraunhofer Center for High Performance Computing: *BeeGFS Documentation – BeeGFS On Demand (BeeOND)* [online], Wiki Page, https://www.beegfs.io/wiki/BeeOND

[24] Kathryn Mohror, Adam Moody, Greg Bronevetsky, and Bronis R. de Supinski: *Detailed Modeling and Evaluation of a Scalable Multilevel Checkpointing System*, in Transactions on Parallel and Distributed Systems, LLNL-JRNL-564721, 25(9):2255-2263, September 2014

[25] Adam Moody, Greg Bronevetsky, Kathryn Mohror, Bronis R. de Supinski: *Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System* in Proceedings of the Supercomputing Conference 2010, New Orleans, LA, November 2010

[26] The Barcelona Supercomputing Center: *The OmpSs Programming Model* [online]

http://pm.bsc.es.

[27] F. Sainz et al.: *Collective offload for heterogeneous clusters*, in Proceedings of the International Conference on High Performance Computing (HiPC), 2015, https://www.bsc.es/ca/printpdf/research-and-development/publications/collective-offload-heterogeneous-clusters

[28] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Jayesh Krishna, Ewing L. Lusk, and Rajeev Thakur: *PMI: A Scalable Parallel Process-Management Interface for Extreme-Scale Systems*, in Recent Advances in the Message Passing Interface: 17th European MPI User's Group Meeting, Springer, Lecture Notes in Computer Science (LNCS), volume 6305, pages 31–41, September 2010, http://dx.doi.org/10.1007/978-3-642-15646-5_4

[29] MPICH Team (Argonne National Laboratory): *Draft of a design of version 2 of the Process Management Interface API*, Web Page, https://wiki.mpich.org/mpich/index.php/PMI_v2_API

[30] Open MPI Team: *PMI Exascale (PMIx)*, Web Page, https://www.open-mpi.org/projects/pmix/

[31] Garrick Staples: *TORQUE Resource Manager*, in Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC'06), ACM New York, article 8, November 2006, http://doi.acm.org/10.1145/1188455.1188464

[32] Morris A. Jette, Andy B. Yoo and Mark Grondona: *SLURM: Simple Linux Utility for Resource Management*, in Proceedings of the 9th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP), Springer, Lecture Notes in Computer Science (LNCS), volume 2862, pages 44–60, http://dx.doi.org/10.1007/10968987_3

[33] SchedMD: *Slurm Workload Manager* [Online], Available: https://slurm.schedmd.com/

[34] D. Krause and P. Thörnig: *JURECA: General-purpose supercomputer at Jülich Supercomputing Centre* in Journal of Large-scale Research Facilities (JLSRF), volume 2, article 62, http://dx.doi.org/10.17815/jlsrf-2-121

[35] A. Lucero: *Simulation of batch scheduling using real production-ready software tools* in Proceedings of the 5th IBERGRID, 2011, https://www.academia.edu/25294817/Simulation_of_batch_scheduling_using_real_production-ready_software_tools

[36] Stephen Trofinoff and Massimo Benini: *Using and Modifying the BSC Slurm Workload Simulator* in Slurm User Group Meeting, 2015, https://slurm.schedmd.com/SLUG15/BSC_Slurm_Workload_Simulator_Enhancements.pdf

[37] Gonzalo P. Rodrigo, Erik Elmroth, Per-Olov Ostberg, and Lavanya Ramakrishnan: *ScSF: A Scheduling Simulation Framework*, in Proceedings of the 21st International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP), 2017, http://jsspp.org/papers17/paper_2.pdf

[38] Walfredo Cirne and Francine Berman: *A Comprehensive Model of the Supercomputer Workload*, in Proceedings of the 4th Annual Workshop on Workload Characterization, 2001, https://doi.org/10.1109/WWC.2001.2

[39] *Ganglia Monitoring System* [Online], Available: http://ganglia.info

[40] *icinga Open Source Monitoring* [Online], Available: http://icinga.com

[41] *RRDtool logging and graphing* [Online], Available: http://oss.oetiker.ch/rrdtool

[42] *DCDB – DataCentre DataBase* [Online], Available: http://gitlab.lrz.de/dcdb

[43] *The Apache Cassandra Database* [Online], Available: http://cassandra.apache.org

[44] *Grafana – The open platform for analytics and monitoring* [Online], Available: http://grafana.com

[45] *Kibana analytics and search dashboard for Elasticsearch* [Online], Available: http://www.elastic.co/products/kibana

[46] *Timelion: The time series composer for Kibana* [Online], Available: http://www.elastic.co/blog/timelion-timeline

[47] *Slurm Scheduler Plugin API*[Online], Available: https://slurm.schedmd.com/schedplugins.html

[48] *Scheduling Configuration Guide*[Online], Available: https://slurm.schedmd.com/sched_config.html

[49] *Heterogeneous Resources and MPMD* Available: https://slurm.schedmd.com/SLUG15/Heterogeneous_Resources_and_MPMD.pdf