H2020-FETHPC-01-2016

# DEEP-EST

# DEEP Extreme Scale Technologies

**Grant Agreement Number: 754304**

# D5.2 – Update

## Software specification

***Final***


**Version:** 2.0
**Author(s):** N. Eicker (JUELICH), Th. Moschny (ParTec), C. Clauß (ParTec)
**Contributor(s):** J. Corbalan (BSC), R. Fischer (BSC), N. Burkhardt (EXTOLL), M. Nüssle (EXTOLL), S. Krempel (ParTec), M. Ott (BAdW-LRZ), D. Tafani (BAdW-LRZ), Z. Ul Huda (JUELICH), I. A. Comprés Ureña (Intel)

**Date:** 31.01.2019

## 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:** | D5.2 – Update |
| | **Deliverable Nature:** | Report |
| | **Deliverable Level:** PU* | **Contractual Date of Delivery:** 30.06.2018 |
| | | **Actual Date of Delivery:** 30.06.2018 Update (v2.0): 31.01.2019 |
| | **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:** Software specification | |
| | **ID:** D5.2 – Update | |
| | **Version:** 2.0 | **Status:** Final |
| | **Available at:** `http://www.deep-projects.eu/` | |
| | **Software Tool:** LaTeX | |
| | **File(s):** DEEP-EST_D5.2_Software_Specification.pdf | |
| **Authorship** | **Written by:** | N. Eicker (JUELICH), Th. Moschny (ParTec), C. Clauß (ParTec) |
| | **Contributors:** | J. Corbalan (BSC), R. Fischer (BSC), N. Burkhardt (EXTOLL), M. Nüssle (EXTOLL), S. Krempel (ParTec), M. Ott (BAdW-LRZ), D. Tafani (BAdW-LRZ), Z. Ul Huda (JUELICH), I. A. Comprés Ureña (Intel) |
| | **Reviewed by:** | A. Auweter (Megware) E. Suarez (JUELICH) |
| | **Approved by:** | BoP/PMT |

# Document Status Sheet

| Version | Date | Status | Comments |
|---|---|---|---|
| 1.0 | 30.06.2018 | Final version | |
| 2.0 | 31.01.2019 | Final version | Update after ESB Review |
| | | | |

# Document Keywords



| | |
|---|---|
| **Keywords:** | DEEP-EST, HPC, Exascale, Software, specifications, job scheduling, resource management, network management, network bridging, system monitoring, Slurm |

# Table of Contents

# List of Figures

# List of Tables

# Executive Summary

This deliverable revisits the requirements for the DEEP-EST system software as they were gathered and presented in D5.1. The initial requirements-analysis resulted in a series of design decisions and choices leading to the hereby described system software stack and its interfaces. The WP5-topics of interconnect management, network bridging, resource management, job scheduling and system monitoring are addressed, bringing the draft sketched in D5.1 into the final specification of the system software that WP5 will implement.

For EXTOLL's novel fabri$^3$ hardware, a revised management software stack has been designed. It builds upon the existing EXTOLL Management Program (EMP) but addresses the new fabri$^3$ hardware design which features integrated management hardware. EMP features such as topology discovery and routing management can now be performed directly by EXTOLL hardware without the need to run EMP on a dedicated network node. The fabri$^3$ design with integrated management will improve significantly the reliability of the EXTOLL fabric in case of PCIe or EXTOLL link failures.

Regarding the bridging of network traffic between the individual modules of the DEEP-EST prototype, the concept and work plan could be defined more precisely following the project's decision on the network technologies to be used in each of the hardware modules: InfiniBand on the Cluster Module, EXTOLL on the Booster Module, and both 40 GE Ethernet and EXTOLL on the Data Analytics Module (see D3.2 for details). Also, among the various options to build the DEEP-EST network federation, an approach has been chosen with three different bridge nodes: one IP bridge between 40 GE and EXTOLL, one IP bridge between 40 GE and InfiniBand, and one MPI + IP bridge between EXTOLL and InfiniBand.

In the field of resource management, recent updates to the Slurm resource management software will help on making Slurm ready for use on the DEEP-EST machine. In particular the support for job packs, which has been added by the Slurm developers to the latest version, addresses the starting of jobs that run across multiple DEEP-EST modules. Further work will have to be done for the allocation of globally shared resources such as DEEP-EST's Network Attached Memory (NAM) and the Global Compute Engine (GCE).

Finally, the system software stack also comprises scalable means for system monitoring. Building on the Data Centre Data Base (DCDB) software that was originally developed in the first DEEP project, further developments have refined and enhanced the feature set for the DEEP-EST monitoring solution. The code base has undergone major cleanups, favouring a modular, plug-in based architecture and DCDB's capabilities to evaluate time series have been extended with the ability to calculate integrals, derivatives and delta series from the time series data. For a comprehensive presentation of the time series of sensor data, a new data connector to the Grafana toolkit will be implemented, so that DCDB time series can be visualised within the Grafana web application.

**Update:** We have added a new section (Section 2.4 "GPGPU Support by EXTOLL network API and Software") to explain the adaptations needed for the new Extreme Scale Booster (ESB) design.

# 1 Introduction

The operation of a Modular Supercomputer requires a system software stack with some additional features to the ones utilised on today's monolithic supercomputers. Deliverable D5.1 presented a detailed analysis of the corresponding requirements in order to set the stage for further work in WP5. Furthermore, a rough design sketch of the software stack was presented. At the time of D5.1, however, only a preliminary and less detailed SW-design could be presented as major HW-design decisions were still pending. A prominent example for this is the decision on the fabric technology to be used in each of the prototype modules. The knowledge of this is a crucial prerequisite in order to provide a detailed plan of the work to be done in Task 5.3 for implementing the Inter-Module Network Bridging.

In the meantime, significant progress has been made on fixing the design and advancing the in-depth architecture of the system software stack. Furthermore, based on the project's design decisions, the work to be conducted in the next 12 months has been planned in detail. It remains to complete the project's goal to provide the system software stack once the hardware of the prototype system is available.

Obviously, many design aspects had to be decided according to the requirements and constraints brought up by the other work packages. Also the implementation of the design cannot be conducted without closely collaborating with them. Furthermore, in the course of fixing the detailed design of the system software stack, several fields were identified that require close collaboration between the individual tasks of this work package and other work packages like WP6. A good example for this is the software layer supporting the Network Attached Memory (NAM) device. On the one hand, this device is designed by WP4 and cooperation with them is therefore naturally required. On the other hand, this software layer will have to interact with the resource management handled in Task 5.4, the Job Scheduler of Task 5.5 and possibly also with ParaStation MPI provided by WP6.

This document presents the design decisions taken since D5.1 and depicts the work plan resulting from them. Similar to its predecessor, this document is organised following the structure of WP5 and its tasks. Each section is structured in a similar way: after shortly revisiting the context and the state of the art, the recent changes and design decisions are presented. We assume the reader is familiar with D5.1 as we will not recap all requirements and decisions here. In addition to that, for each task – and therefore for each component to be developed in the work-package – a high-level design specification is presented.

# 2 Interconnect Management

The Modular Supercomputer Architecture consists of multiple different partitions. The partitions use different interconnection networks, which are chosen based on the requirements of each partition. This chapter focuses on the interconnection network of the Extreme Scale Booster and describes the management software for the EXTOLL fabric.

## 2.1 Revisiting Context and State of the Art

The interconnection network for DEEP-EST is based on fabri$^3$. Instead of attaching the network interface directly to a compute node, the network interface is integrated into fabri$^3$. The compute nodes itself are connected to fabri$^3$ by PCIe links. This improves the reliability of the direct network, as the compute nodes and the interconnection network operate in different power domains. Shutting down a compute node does not result in an irregular topology, compared to a network interface card for a direct network, which is directly attached to a compute node. fabri$^3$ integrates multiple EXTOLL Tourmalet networking ASICs to form a 3D-Mesh network topology. Network links not used by the mesh are available on cable connectors, either to close the rings in the different dimensions to form a 3D-Torus, or for connecting multiple fabri$^3$ instances to form a larger network. More details on the hardware implementation of fabri$^3$ are given in D4.3.

In order to use a fabri$^3$ network, the interconnection network must be configured. This task is fulfilled by the interconnect management software. Not only does the management software have to configure each single fabri$^3$ instance, it also has to configure the interconnections between different fabri$^3$ instances.

The basic functionality for configuring an EXTOLL-based interconnection network is provided by the EXTOLL management process (EMP). EMP was already used for the configuration of the DEEP and DEEP-ER prototypes.

The existing EMP provides the following features to configure the interconnection network:

- network topology discovery
- routing calculation for unicast and multicast traffic including support for hierarchical and inter-connected topologies in complex networks
- monitoring of basic network parameters
- ability to stop and resume network traffic globally, to schedule hardware interventions
- simple command line interface
- RESTful API
- web-based front end

The EMP software suite is built from different components: the master and slave daemons. The master runs on one compute node in the interconnection network. It detects and configures the network by a direct, PCIe-based access to the network. The slave daemons run on each

compute node and provide an EXTOLL network independent access to the network hardware, which does not rely on accessing remote nodes directly via the EXTOLL fabric. The slave daemons communicate with the master daemon via TCP/IP. The slave daemons are mainly used for basic network monitoring.

## 2.2 Recent Changes and Design Decisions

This section provides an overview of the recent changes and design decisions for the interconnect management software. For the EXTOLL interconnect management, the existing EMP will be adapted and extended to support the new fabri$^3$ hardware and provide enhanced resiliency features. The architecture of the management software is depicted in Figure 1.

One of the main changes for EMP is how the network hardware is accessed. Before fabri$^3$, the EMP master daemon configured the network by accessing the network hardware directly via PCIe. One goal for fabri$^3$ is to separate network management from the compute nodes. The compute nodes are only connected to fabri$^3$ via PCIe. The network management software must be able to configure the network even if no connected compute node is running. Therefore, fabri$^3$ includes management hardware, which provides configuration access to the EXTOLL Tourmalet ASICs inside fabri$^3$via I$^2$C. The management hardware consists of multiple micro-controllers. Each micro-controller has access to 4 Tourmalet chips via I$^2$C. The micro-controllers themselves are connected to a management controller by USB. For a detailed description of the management hardware for fabri$^3$ see D4.3.

To reflect the changes in the management hardware, the architecture of EMP was revised. The task of accessing the network hardware was moved from the master daemon into the slave daemons. A slave daemon runs on each management controller. The slave daemon accesses the network hardware by sending read and write requests to the micro-controllers via USB. The micro-controllers translate these requests into I$^2$C accesses to the network hardware. Responses from the network hardware are forwarded back to the slave daemon by the micro-controller. The communication between the master and slave daemon is realized by a TCP/IP socket. This enables a flexible architecture for the interconnection management software. The master daemon can run on any node having IP management network access to the slave node. This can either be any fabri$^3$ management controller, or another dedicated management node. For now, it is planned to select one fabri$^3$ management controller as master daemon. The proposed architecture also makes a redundant master daemon setup possible.

As the I$^2$C access to the network hardware is slow compared to the speed of the USB interface, the micro-controller firmware that will be developed has to integrate some more functionality to hide the interface speed differences. Therefore, the slave daemon must be able to issue multiple I$^2$C requests to different Tourmalets in parallel. Furthermore, for monitoring the network hardware, the micro-controllers will gather the data on their own and store it for later processing by the slave daemon. The command protocol for the slave daemon to micro-controller communication will include commands to start and stop the data collection, as well as for defining which data should be collected. The collected data can e.g. include network performance counters like processed packets or used credits to trace network traffic and to detect traffic hot spots. Another important functionality that will be included is a notification mechanism, enabling the

micro-controller to inform the slave daemon about fault conditions of the network hardware.

### 2.2.1 EXTOLL network resiliency

Another goal for the interconnection management software is to improve the reliability of the network in the case of an error event. Two main causes for network faults were identified and solutions for handling these faults were developed. The first class of faults is the failure of a PCIe link between fabri$^3$ and a directly connected compute node. This can happen either because of rebooting or shutting down a compute node for maintenance, or because of a hardware failure. The experiences with the DEEP-ER prototype show that these kinds of faults can be considered the main cause of network failures in a direct network. The second class of faults are link errors caused by defective or miss-plugged cables. As for fabri$^3$, the majority of link connections are done via PCB, the likelihood for link defects decreases compared to non fabri$^3$ systems. Nevertheless, the remaining cables for inter-fabri$^3$ connections are vulnerable to faults, though the connections are improved by using a newly developed cable type. For a description of the new cables see D4.3.

For handling PCIe link faults the following strategy was developed. In the case of a failing PCIe link, the target node can no longer remove the packets causing the network's flow control mechanism to block further network traffic. From a network perspective it is more important to keep the network alive than waiting until a host may become available again. Therefore, it was decided to drop network packets in Tourmalet with the failing PCIe link until the PCIe link becomes available again. Activation and deactivation of the packet dropping will be initiated by the interconnection network management software when a failing PCIe link is detected. If the PCIe link of a node fails, the management micro-controller gets notified by an interrupt, which then starts the packet dropping. The interrupt is also forwarded to the master daemon for monitoring purposes. The packet dropping gets deactivated, as soon as the PCIe link becomes available again.

The handling of link failures cannot be performed in a local management instance, as the routing tables of nodes not directly connected to the failing link might need to be rewritten for an overall deadlock-free routing. Therefore, the information that a link failed is forwarded to the master daemon, which then has to decide how to adjust the routing of the network. The target network topology for the DEEP-EST ESB is a 3D-Torus. This regular topology provides no implicit resilience so that the standard routing algorithms cannot be applied in case of a failing link. The management software then tries to make some local adjustments to the routing to maintain a global deadlock-free routing in the network. If only one link of a ring in the torus fails, the routing direction in this ring is limited to only one direction. If more than one link in a ring fails, the master daemon tries to establish deadlock-free routing by adding a detour around this link. As last resort, if no deadlock-free routing is possible anymore by local adjustments to the torus routing, up*/down* routing is considered as a deadlock-free alternative. Up*/down* routing guarantees a deadlock-free routing for any irregular topology, but is not as efficient as the torus routing.

## 2.3 High-level Design Specifications

The interconnection management software has to fulfil several tasks as described in the previous sections. Beside the configuration of the network, it also has to monitor the network and react to network faults in order to keep the interconnection network alive and functional.

Based on the existing management software for EXTOLL and the DEEP-ER system, the interconnection management software for DEEP-EST will be constructed as follows. The central management component for the interconnection network is formed by a master daemon. It sets up the network and re-configures it in case of network faults. The master daemon does not directly access the hardware, as in previous versions. Therefore, it is independent from the network hardware and can run on any node with IP access to the slave daemons. For convenience, the master daemons run on one selected fabri$^3$ management controller.

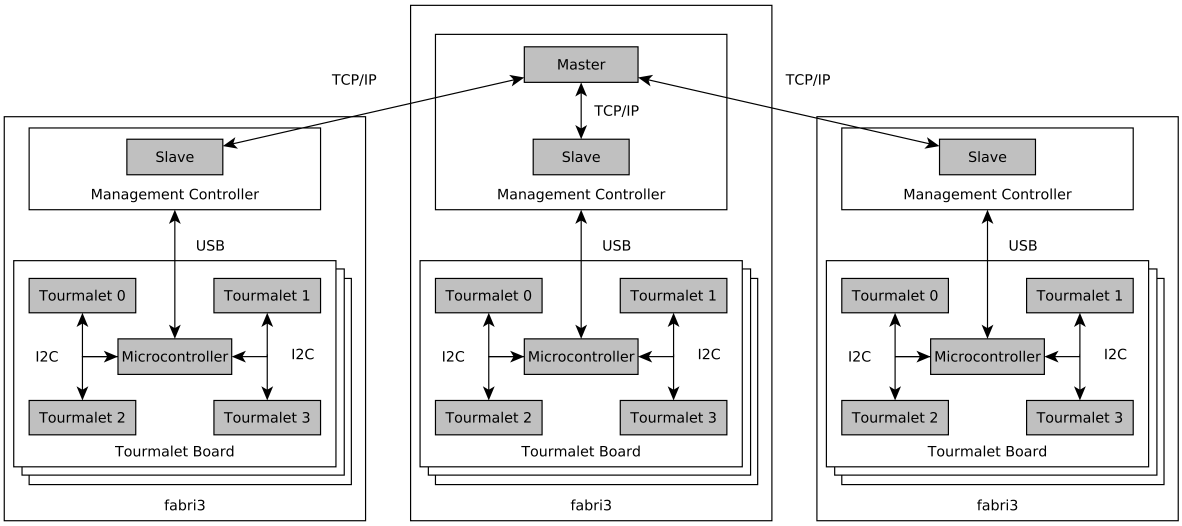


Figure 1: fabri$^3$ management software architecture

The slave daemons provide the network hardware access for the master daemon. The communication between master and slave daemons is done by TCP/IP. On each fabri$^3$ instance one slave daemon runs on the integrated management controller. Each slave daemon is connected to 16 micro-controllers integrated into the fabri$^3$ Tourmalet boards. The slave daemon forwards received network hardware accesses from the master daemon to the micro-controller attached to the Tourmalet to which the master daemon needs access. The micro-controller firmware translates the requests into I$^2$C transactions, which accesses the network hardware.

The master daemon will provide different user interfaces. All interfaces provide the same level of control for the interconnection network. First, there is a web-based interface. It provides a graphical user interface for an easy-to-use network topology visualization. The second interface is based on a RESTful API. This interface enables a comfortable way for scripting network setup and monitoring tasks. The third interface is a command line client. This client is designed as a wrapper for the RESTful API.

As described in the previous section, two classes of network faults were identified that should

be handled by the interconnection management software. The first and main class of faults are failing PCIe links between fabri[3] and a connected compute node. To prevent a network blockage, packets targeted to the failed compute node are dropped in the destination Tourmalet network chip. The interconnection management software monitors the PCIe links and initiates in the case of a link failure the packet dropping. After the re-connection of a PCIe link, the management software disables the packet dropping again.

The second fault class are failing network links. The likelihood of failing network links can be considered less than failing PCIe links. Nevertheless, failing network links are more complicated to handle. Therefore, the information about a failing network link is propagated to the master daemon, which decides about the fault resolution. The Tourmalet network chip triggers an interrupt when it detects a network link failure. The interrupt is caught by the management micro-controller, which starts the propagation of the fault to the master daemon. The master daemon is then responsible for deploying a new routing to the network, which ignores the failing network link. As the network topology becomes irregular when at least one link fails, the master has to ensure that the new routing is still deadlock-free. There are different approaches to maintain a deadlock-free routing. The target fabri[3] network topology for DEEP-EST is a 3D-Torus. If only one link in a ring of the torus fails, the routing direction on the failing ring is limited. If more than one link in a ring fails, detours around this failing link are established. If it is not possible to add detours without introducing a routing deadlock, up*/down* routing is used to provide a deadlock-free routing on an irregular topology.

## 2.4 GPGPU Support by EXTOLL network API and Software

**Update:** The DEEP-EST ESB design, as introduced with the review at M18, will be based on GPGPU accelerators providing the compute power. NVIDIA V100 GPUs were selected on the hardware side. This implies, that the NVIDIA driver stack will be used to access the GPUs. It is mandatory to support high-bandwidth and low-latency access to memory used by the code running on the GPUs. For this purpose, the EXTOLL driver stack will be extended to support the latest NVIDIA GPUDirect for RDMA API specification, which will be extensively used for the ESB.

GPUDirect for RDMA allows mapping of (internal) GPU memory to the Base Address Region (BAR) of the GPU visible on the PCIe system of the node. V100 GPUs will be able to map all (or almost all) of its internal memory to its BAR. The EXTOLL software is able to map such BAR regions to be used as source or destination of RMA operations (put and get) already. This feature was introduced in the course of the original DEEP project. GPUDirect now offers a supported way of implementing direct access to accelerator memory as seen from the accelerator side.

Supporting direct remote access to accelerator memory will be crucial to be able to reach the maximum physically supported communication bandwidth from/to the accelerator. In cooperation with other tasks in WP5 and WP6, it will be analysed how communication over EXTOLL can be optimally implemented for example by the ParaStation MPI implementation. While zero-copy direct RMA transfers are seen as the optimum for larger messages, other means could be exploited for small messages. One technique that will be analysed is to map GPGPU

buffers into user-space (exploiting again the GPUDirect API) and using CPU-driven memory copy to move the data of an MPI operation into the GPU's memory instead of relying on the CUDA-programmed, relatively high-latency DMA operations.

Furthermore, it will be analysed how the GCE can be effectively exploited by the ESB software stack.

# 3 Inter-Module Network Bridging

This section presents and discusses the latest evolution regarding the system software for the Network Federation (NF) and the related network bridging between different modules of the Modular Supercomputer Architecture (MSA). In doing so, this section considers itself an update to Section 3 of Deliverable D5.1 and therefore primarily builds upon the content already presented there.

## 3.1 Revisiting Context and State of the Art

Starting point for the network bridging in DEEP-EST is the capability of the *pscom* communication library [11] to drive different network technologies simultaneously by means of so-called *pscom plugins*. That way, nodes equipped with two (or even more) network interfaces can act as *gateways* between different domains of interconnect technologies and hence between different modules of the MSA.

As discussed in D5.1, there are several architectural approaches conceivable for integrating such *gateway nodes* into the MSA modules and for routing the network traffic between them. However, a quite obvious approach is to deploy a set of dedicated nodes that constitute an overlap between the different technologies in such a way that only one network translation is required for sending a message from one module to another.

According to this, on all of these gateway nodes likewise dedicated *gateway daemons* are started and serve as forwarders for the messages to be sent from one network domain into the other. If two processes are located in different MSA modules with different network technologies, they use a designated *gateway plugin* for the communication between them, 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 doing so, a *logical* connection can be established between them on the basis of a *store-and-forward* mechanism for message chunks to be passed along by the respective daemon.

A production environment where this gateway concept has already been implemented is the JURECA system at the Jülich Supercomputing Centre. This system features an Intel Haswell-based general-purpose Cluster of 1882 nodes equipped with InfiniBand (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 this setup, 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) to forward the message chunks.

Besides the need for bridging MPI traffic, the requirement analysis conducted in advance of D1.1 revealed the necessity of having access to the Network Attached Memory (NAM) modules from everywhere in the DEEP-EST prototype. For this purpose, NAM modules can be attached to any EXTOLL link in the Network Federation [12], as it has also already been discussed in D5.1. By utilizing the *libNAM* library, applications are able to allocate memory regions on the NAMs via a central NAM manager. The actual memory accesses towards the NAMs can then

be performed by means of *Put* and *Get* functions provided by the API of the libNAM. However, as the libNAM is based on the EXTOLL RMA capabilities and the related low-level *librma2* library, direct NAM accesses are only possible from nodes within the EXTOLL fabric.

## 3.2 Recent Changes and Design Decisions

By now, most of the design decisions concerning the hardware architecture of the different modules of the DEEP-EST prototype have been made and also the module-internal interconnect technologies have been selected. However, in particular the selection process for the network fabric to be used inside the Cluster Module (CM) has taken longer than anticipated since thorough benchmarking was needed to compare InfiniBand EDR and Intel Omni-Path. The final results were in favour of InfiniBand (see Deliverable D3.2). Regarding the Extreme Scale Booster (ESB), EXTOLL will be the interconnect technology of choice, which will also be used for the Data Analytics Module (DAM), but here together with high-speed Ethernet (40 GE).

Furthermore, it has also already been decided that the prototype will go for a solution with *three* intersection points between the different networking technologies, as it is here also denoted in Figure 2:

- One IP bridge between 40 GE and EXTOLL
- One IP bridge between 40 GE and InfiniBand
- One MPI + IP bridge between EXTOLL and InfiniBand

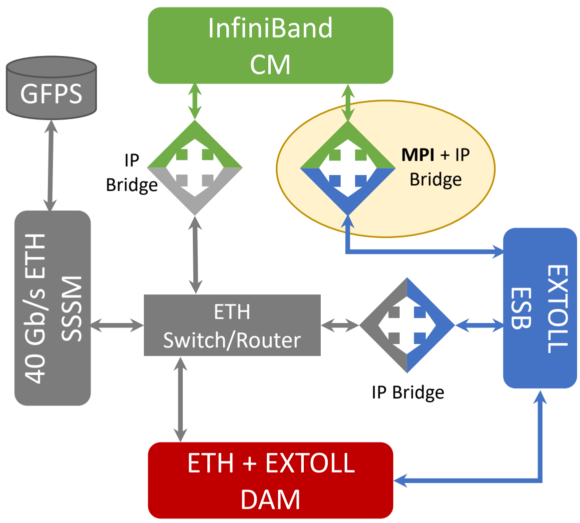


Figure 2: Solution for the Network Federation with *three* bridging points.

Based on preliminary assessment of the application requirements, we identified the highest performance requirements in the MPI bridge between the EXTOLL domain of ESB plus DAM and the InfiniBand network in the CM. In contrast to a previously discussed solution, where only two bridging points were foreseen (one between CM and 40 GE plus one between ESB and

40 GE), the chosen solution features the advantage that only *one* network translation is required for MPI messages between ESB/DAM and CM. According to this, the designated solution for the prototype actually conforms to that version of a NF where dedicated gateway nodes *directly* bridge between each pair of MSA modules (if necessary), as it has been introduced in D5.1 and as shown again in Figure 3.

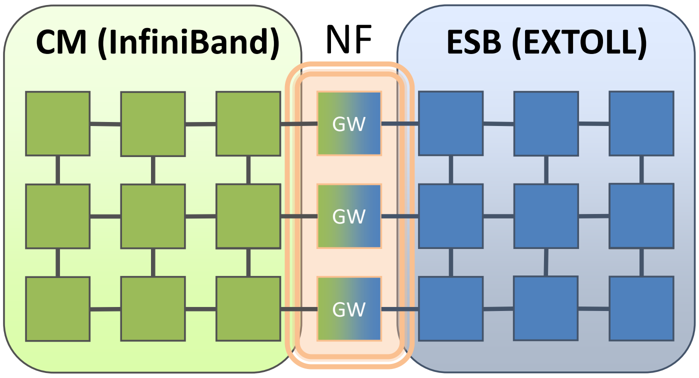


Figure 3: Designated MPI network bridging via gateway nodes (GW) between ESB and CM

Regarding the NAM, a new libNAM version (v2) is currently in the specification stage within Task 4.5 (see Section 3.5 in Deliverable D4.3). A central and important change for the libNAM API will be the possibility to access the same NAM regions from different processes within the system. For this purpose, the NAM manager has to generate globally valid allocation identifiers that can be passed around between the processes. However, since this possibility has already been presumed in the earlier planning stages of WP5 (as well as in those of WP6, here especially in respect of an MPI wrapper as planned in Task 6.1), these design decisions do not undermine but rather substantiate the design specifications as drafted in D5.1. Therefore, the essential point stays the same: While the NAM manager can be contacted from any point in the MSA via TCP, accesses from modules *not* equipped with EXTOLL (i.e. the CM) will either be impossible, or – for which this task is striving for – some forwarding mechanism needs to be implemented.

## 3.3 High-level Design Specifications

As the actual manifestation of the later NF is almost determined by now (merely the number of gateway nodes is yet to be decided), the envisioned software architecture for the MPI inter-module bridging framework has become clearer as well. According to the assumptions made in D5.1, the actual framework will correspond to the layer model as depicted in Figure 4. As one can see, the two MSA modules – the left one using EXTOLL, here represented by the ESB, and the CM on the right side using InfiniBand – are bridged by a gateway daemon running on a gateway node.

Naturally, there will be a demand for having more than just one gateway node between the distinct modules. However, concerning the eventual number of gateway nodes, there are two different aspects to be considered: on one hand, from the hardware point of view, the number

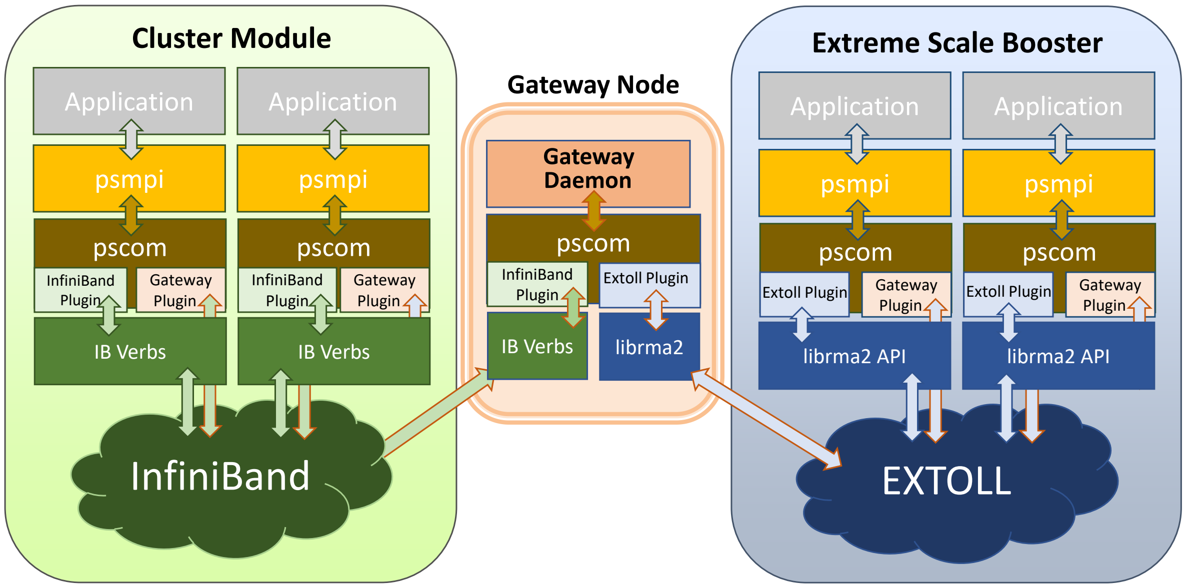


Figure 4: The planned daemon-based gateway concept for the network bridging

should be chosen in accordance with the expected MPI traffic between them – which in turn obviously depends on the DEEP-EST applications and their communication patterns. Therefore, a good estimate has to be done for the prototype development in order to mitigate the inter-module bottleneck while not wasting project budged for later unused resources. In fact, for doing so, the project will acquire three *gateway evaluator nodes* to get a clearer understanding of this topic (see Deliverable D4.3). On the other hand, from the software point of view, the MPI bridging framework should be capable of dealing with multiple gateway nodes in such a manner that at least some static load balancing can be achieved. For this purpose, we envisage – at least in the first instance – a static routing approach where an appropriate load balancing scheme is hard-coded, for example, in terms of a routing file to be generated either manually by the system administrator or automatically perhaps by the resource manager.

According to this approach, the static routing file contains an appropriate mapping between address spaces in terms of IP addresses and subnet masks, respectively. During the connection establishment between two MPI processes in different modules, this mapping information will be evaluated for choosing the right, common gateway daemon. After connecting to this daemon for the first time – be it either according to an *on-demand* mechanism or right at session startup – a logical connection between the two processes via the daemon has been established: messages to be sent, for example, from the CM to the ESB will then initially be passed via InfiniBand to the respective gateway daemon, which stores and instantly forwards the messages via EXTOLL to the actual receiver.

As the number of gateway nodes and their daemons is expected to be much smaller than the number of compute nodes and MPI processes per module, the mapping scheme naturally enforces a *multiplexing* of several logical connections through each gateway. At this point the question arises, whether the gateway daemons should be system daemons to be started at boot time or if they should rather be user-assigned, which means that they would be started

upon a user-issued request for initiating a parallel MPI session across multiple modules. Currently, we assume that the *batch system* – as it will be adapted in Task 5.4 and Task 5.5 to the requirements of the DEEP-EST prototype – will be responsible for starting the gateway daemons appropriately according to the user's and/or the application's demand. Of course, this approach implies that the gateway daemons are user-assigned, but organizing the gateway nodes as shared resources would still be possible by co-locating the daemons of different users (if allowed by the respective scheduling policy).

Regarding the actual implementation of the forwarder mechanism within the daemons, we plan as a first step a quite generic architecture that should in principle be able to translate between any network architecture by applying a plain store-and-forward routing. However, as a next step, we also plan to realise performance improvements, for example, by utilizing shared intermediate buffers for avoiding unnecessary copy operations on the gateway nodes and by applying a more sophisticated pipelining scheme for gaining better data throughput. In addition, some more interconnect-specific optimizations – hence tailored to the interplay between the pair of EXTOLL and InfiniBand – are also envisaged.

With respect to the global accessibility of the NAM nodes, we currently see no need for major updates or changes regarding the design and specification as it has been presented already in D5.1. Therefore, we just summarise the related architectural draft here. According to the draft of this part of the inter-module bridging framework, invocations of libNAM functions like *Put* and *Get* that are issued from within a module not being part of the EXTOLL fabric (i.e. from within the CM of the prototype) need to be intercepted and redirected to a respective forwarding entity running on the gateway nodes. These forwarding entities, which are actually quite similar to the gateway daemons described before, have the responsibility to receive such requests from the CM side via higher-level network messages transferred through InfiniBand, and to transform them into libNAM or even more low-level librma2 requests then to be issued towards the actual NAM. On the other side, NAM requests already satisfied by the EXTOLL network towards the gateway nodes have to be encapsulated again into regular network messages to be sent via InfiniBand to the processes that invoked the access functions. As one can see, this approach would actually allow to access the NAMs from everywhere within the DEEP-EST prototype but just with a limited performance for the CM. Therefore, additional optimizations are envisaged that should help to improve at least the achievable data throughput between CM nodes and NAMs, for example, by applying again an appropriate pipelining scheme.

# 4 Resource Management

## 4.1 Revisiting Context and State of the Art

As described in detail in D5.1, we consider the Resource Management to consist of two parts, the *Resource Allocator* and the *Process Manager*. The Resource Allocator has two main tasks: first is to take the requests from the users and to pass them over to the Job Scheduler; second is to take the decisions of the Job Scheduler and trigger the Process Manager to prepare the correct resources and start the users' processes in the right way on the right resources. The Process Manager launches and controls the processes, acts as connection broker for MPI processes, and handles I/O forwarding between the user and the processes.

In the DEEP-EST project, Slurm will be used as Resource Allocator and the ParaStation Management will be used as Process Manager. The connection of the two is made by *psslurm*, a plugin to the ParaStation Management Daemon replacing *slurmd*, which runs on each node in vanilla Slurm.

## 4.2 Revisiting System Software Requirements

### Handling Heterogeneous Resources

An increasing number of applications is using heterogeneous compute resources. This might be either within a single execution entity, i.e. a single MPI job, or throughout the different tasks within the workflow that implements the application. Therefore, the Resource Management System needs to be able to manage these kinds of resources and provide them to the jobs. The mentioned resources include different types of processing elements, novel memory class devices and their capacity, or association to the storage system. D5.1 provides more details and a few examples. If the resources reside in different modules that are connected via special gateways or similar constructs (see Section 3), the Resource Management System has to manage these gateways as a kind of global resource, too. This generally holds for all resources that are not represented by regular MPI processes. In summary, it has to provide a flexible *horizontal* (spatial) diversity of resources inside one job step (see Figure 5).

In Slurm the canonical way to express tasks in a workflow is to use *job steps*. If applications want to use different sets of resources in different tasks of their workflows, it has to be possible to change the resource allocations not only between jobs, but in some way even between job steps, providing a *vertical* (temporal) diversity of resources (see Figure 5).

With release 17.11, Slurm recently introduced a feature called *job packs*. It enables allocating different kinds of resources at once and running multiple executables as one job step at once (see also Section 5.1). This is needed for running heterogeneous jobs, e.g. jobs using nodes of two different node-types.

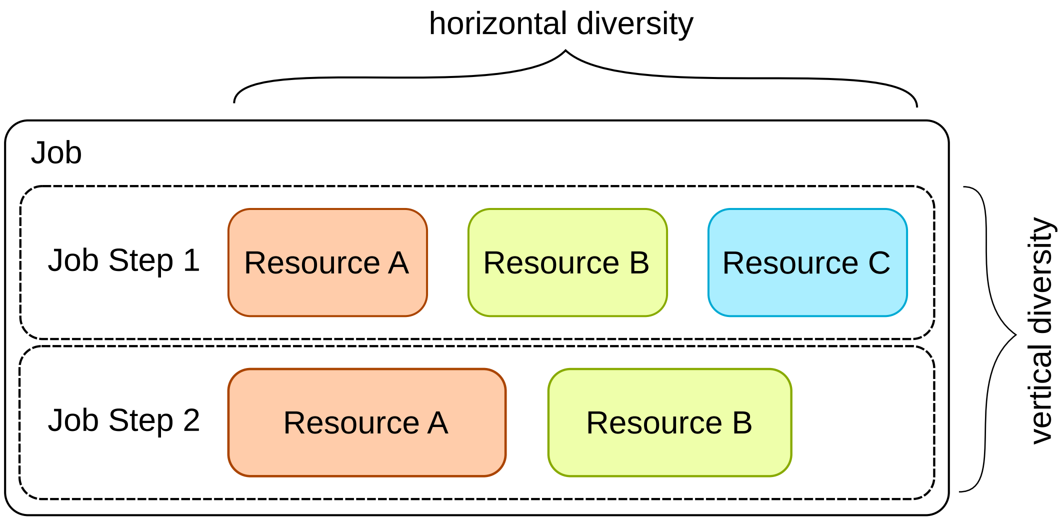


Figure 5: Horizontal and vertical resource diversity

#### Handling Global Resources

One of the project's topics is the support of global resources: resources that are globally accessible, limited and shared. The DEEP-EST prototype will provide two types of such global resources: the Network Attached Memory (NAM) and the Global Collective Engine (GCE). The NAM is fast memory attached to the EXTOLL network and managed by one central NAM manager. It can be accessed from processes on each node inside the network (see Section 3.5 in Deliverable D4.3). The GCE is a computing device for collective operations. It is also attached to the EXTOLL network and will have similar accessibility as the NAM.

Currently there is no support at all for such kind of resources in Slurm.

## 4.3 Recent Changes and Design Decisions

#### Handling Heterogeneous Resources

The full support of Slurm's *job packs* feature has already been integrated into psslurm and is currently in the testing phase. That means it is now possible to start all MPI processes of a job pack in a single `MPI_COMM_WORLD`. These MPI processes may have different executables executed on different sets of nodes. For example, this is needed if a compute cluster has some kind of booster nodes using a different micro architecture. As the current Slurm version was the first to introduce the *job packs* feature, it is anticipated that this feature will undergo some slight changes in the next Slurm release, refining and further enhancing support for heterogeneous workflows. Therefore some further adaptations for psslurm might also be needed. However, the feature is expected to stabilize afterwards limiting the amount of work on the psslurm side to general maintenance level. All in all, with supporting job packs a big step is made to supporting horizontal (spatial) diversity (see Figure 5).

### Handling Global Resources

For the handling of global resources, emphasis will be put on implementing the required functionality for Network Attached Memory first before adapting the solution to other kinds of resources such as the Global Collective Engine.

It remains an open point of discussion, whether the scheduler should have the complete control over the whole NAM so that allocations will have to be made before the job's start by the Resource Allocator as requested by the user, or if the user should have the opportunity to allocate further NAM space from inside the program. In the first case, persistence of the memory over job boundaries will be a problem to resolve and NAM space would be blocked for the whole runtime of a job, even if it is not needed all the time. On the other hand, if users are allowed to allocate additional NAM space themselves directly through the NAM manager, it would need to provide more high-level functionality, such as reservations for the future, to enable the scheduler to do its job, and quotas to prevent abuse. Partitioning the NAM so that a portion of it can be used freely and dynamically, i.e. in a file system-like manner, while the other portion is completely controlled by Slurm as described above, could be a solution satisfying both use cases.

# 5 Job Scheduler

## 5.1 Revisiting Context and State of the Art

The job scheduler context stated in D5.1 is the reference for this deliverable. The DEEP-EST job scheduler is going to be based on the Slurm scheduler [13], extending and complementing the new concept of heterogeneous jobs appeared in version 17.11 (also called *job packs* in Slurm [15]). Currently, more information on Slurm's handling of heterogeneous jobs is available, extending what could be found when the first version of this deliverable was released [16]. The heterogeneous jobs feature allows the submission of a single job composed of multiple resource allocations, each one running a different "component", which can potentially be a different binary. Slurm is aware that subsequent components belong to the same heterogeneous job, and therefore, some options are propagated simplifying such process for the system user. Since each component is a resource allocation, the same available options for configuring non-heterogeneous jobs are also valid for each individual component.

The integration of MPI jobs with this new concept allows to start multiple components in a single `MPI_COMM_WORLD` or in independent `MPI_COMM_WORLD` communicators.

The following examples are based on the Slurm web page examples but have been updated to reflect DEEP-EST characteristics.

```
$cat my_job.cmd
#!/bin/bash
#SBATCH --mem-per-cpu=16g -N1 -C skylake
#SBATCH packjob
#SBATCH  --N32 --exclusive -C NAM --mem_nam=100GB
srun master &
srun --pack-group=1 slave &
wait
$sbatch my_job.cmb
```

This first case creates a heterogeneous job composed of a first allocation requesting 1 node, 16 GB of memory per CPU and, as constraint, nodes with "skylake" feature. The second allocation is requesting for 32 nodes, and 100 GB of NAM memory (this is a new feature for Slurm for which the syntax has not been decided yet.)

```
$cat my_job_single_comm.cmd
#!/bin/bash
#SBATCH --mem-per-cpu=16g -N1 -
#SBATCH packjob
#SBATCH  --N32 --exclusive -C NAM --mem_nam=100GB
srun master : slave &
wait
$sbatch my_job.cmb
```

In this second example, the same allocation and distribution is created, but master and slave are executed sharing `MPI_COMM_WORLD`, allowing them to communicate through MPI, thus alleviating the need for using the file system or other mechanisms for data exchange.

## 5.2 High-level Design Specifications

Three main requirements must be covered by the job scheduler:

**Efficient job scheduling for modular architecture** In previous DEEP projects, the focus was on efficient execution of individual applications. However, when running in production environment, there are other factors that affect the application response time rather than only the execution time. For instance, the *wait time*, which depends on: the system load, the application requirements, and the application's flexibility to be started on different resources. However, for many users, when the same application can be executed in different modules, it can be complicated to provide resource allocation details such as number of nodes, requested memory or CPUs per task. We will work on scheduling policies that estimate the requirements in the different modules based on (1) historic information (if available), and (2) extrapolating requirements for one module to other modules. We will also consider the utilisation of algorithmic-based application costs already used in the context of individual applications to minimise the amount of information that is requested from users [14].

**Efficient support for coupled workflows using DEEP-EST features** Applications requiring a workflow sometimes need to transfer huge amounts of data from one sub-job to its dependent sub-job. In the above scenario, all the data need to be stored in some type of secondary memory (like disks) at the end of the first sub-job and read again into the main memory at the beginning of the dependent sub-job. This can be a very time-expensive operation considering the amount of data and the frequency of such operations. To overcome such a problem, we propose a new feature in Slurm that guarantees some overlap of the execution of two sub-jobs having such type of dependency. This overlapping enables the introduction of new functionalities to directly transfer data between consecutive parts of the workflow, eliminating the intermediate step of saving it on external storage.

**Efficient scheduling and management for shared global resources** In DEEP-ER, the Network Attached Memory (NAM) was introduced. However, when running in a shared system, the utilisation of this global resource opens many questions such as the implementation of per-user quotas or the life time of global NAM handlers. Apart from the NAM, we consider including the management of other global shared resources, too.

### 5.2.1 Efficient Job scheduling for modular architectures

Slurm scheduling is divided into two areas that cooperate with each other: The scheduler and the resource selector. In Slurm they can be dynamically configured using a plugin mechanism. The scheduler is in charge of dealing with the job queue, to ensure priorities (as a general concept), deal with reservations, etc. The resource selection (or select plugin) is the entity aware of the resources available per node and the status of each node as well. It acts like a filter, evaluating job resource requirements and considering if a job can be immediately started or not, based on resource availability.

The default scheduler in Slurm is *backfilling*. In that case, jobs are potentially considered for execution in two cases:

- Being the first job in the queue. In that case, the job is considered for execution by the FIFO thread and the job is the one with the highest priority.
- Not being the first job in the queue. In that case, the job is considered for execution by the backfilling thread and the job is not the one with the highest priority.

In the first case, if the resource selection plugin evaluates that the job can be started, no other conditions (apart from limits evaluated at submission time) need to be considered and the job can be started immediately. In the second case, even though there are resources available, there is a second precondition for starting the job: the job cannot delay the $N$ previous jobs in the queue. This evaluation is done based on the wall-clock time requested by the user.

In DEEP-EST we plan to create a new plugin for resource selection derived from the already consolidated `cons_res` plugin. The `cons_res` plugin is a very powerful mechanism for considering different configurations (more details can be found on the Slurm web page [17]):

- Computational resources (`CR_CPU`, `CR_Board` ,`CR_Socket`, or `CR_Core`):
  CPU, Baseboard, Socket, or Core as a consumable resource.
- Memory resources (`CR_Memory`, `CR_Socket_Memory`, `CR_Core_Memory`, `CR_CPU_Memory`):
  Only memory, socket and memory, core and memory, or CPU and memory as a consumable resource.

When using this plugin, nodes must include the amount of "resources" available per node, in order to allow the node selection plugin to evaluate which nodes are available (or not) for resource selection. Apart from consumable resources, nodes can be tagged with "features" in the `slurm.conf` file. These features are used like filters by the `--constraint=list` srun option (see `sbatch` [19]/`srun` [18] web page). Features are typically used to describe node characteristics existing in some of them. For instance, we can envision a node description like this one:

```
NodeName=cm01[0-50] CPUs=24 Sockets=2 CoresPerSocket=12
  ThreadsPerCore=2 RealMemory=196608 State=UNKNOWN
  Feature=skylake,OPA,SSD

NodeName=dam01[0-20] CPUs=40 Sockets=2 CoresPerSocket=20
  ThreadsPerCore=2 RealMemory=393216 State=UNKNOWN
  Feature=cascadelake,NAM,SSD,EXTOLL
```

Given these characteristics, a user can select nodes in the Cluster Module or the DAM by just setting `--constraint=skylake` (to select the cluster module) or `--constraint=cascadelake` to select the DAM.

Given that, on a modular architecture, a job could potentially run in more than one module at the same time, or can be supported to run in different nodes (mainly if they are binary compatible). Our target is to provide a smart scheduler, aware of modules, where users will have to specify resource requirements only once. Thereafter, the scheduler will extrapolate the list of resource requirements (along with the list of modules where the job can run) to the required resources in the different modules. Enforcing the user to specify different resource allocations, including

wall-clock limit, for different modules, is not suitable when thinking in a machine with, potentially, many modules.

The idea is to extend submission options to specify scheduling hints. We will target that problem in different phases. First, we will identify the main steps involved in job scheduling and resource selection and we will create a new select plugin based on the `cons_res` where the plugin will check resources in different nodes.

The new plugin will be able to work with two types of potential allocations:

- In cases when the application only runs in one module, but can potentially run on different modules: The new select plugin will return one list where each element will be in turn a list of nodes, all of them belonging to the same module. The scheduler will have to select the best module based on the estimated slowdown. It must be noted that, even though one job can be faster in one particular module, in case it is busy, it should be executed on another available module, resulting in a better selection of a "slower" yet free module.
- In cases when the application needs more than one module to run: In that case, the new select plugin will return a list with pre-selected nodes and the scheduler will select a combination of both lists, the one that reports the lowest wait time.

We will attack this goal in different phases: First we will target the problem of resource selection basic functionality for a modular architecture assuming the user provides the requirements.

- New DEEP-EST select plugin supporting resource requirements for modular architectures based on user-provided requirements.
- New (extension of the existing one) "best nodes" node selection algorithm taking into account the modular architecture.

Second, we will target the problem of resource estimation. Based on a simplified resource specification and some scheduling hints, the scheduler will create a detailed list of resource requirements for the cluster. The algorithms and models used to, for example, estimate wall-clock limits, will be improved based on historical information. The scheduler will take into account (differentiating between) user-provided information and system-estimated requirements when considering critical decisions such as killing the job when job runtime exceeds the wall-clock. Slurm includes in `slurm.conf` an option to specify a grace time for jobs exceeding the wall-clock time. For those cases when the wall-clock time has been estimated, this action must be carefully considered.

Finally, we will implement a new scheduling hint tag to provide an abstraction of the execution cost of the application based on the main application algorithm. This approach has been applied at a lower granularity with success and we plan to include it as a mechanism to automatically provide runtime estimations independently of the architecture.

### 5.2.2 Efficient support for coupled workflows using DEEP-EST features

We did a detailed investigation of the job packs support in Slurm and came up with the following findings:

- A whole job pack is tested for the validity of resources requested by each job step in it.

The whole job pack is rejected even if the requested resources of a single job step are invalid.

- All of the job steps in a job pack are allocated with the resources at the same time and all of these job steps run in parallel to each other.
- A whole job pack is put on hold if resources requested by any of the job steps are not available until they become available.

All of the above features of the job packs support in Slurm are found to fulfil most of the needs of our applications. All of the applications requiring different job steps running on different resources in parallel can work out of the box. However, some applications requiring workflow type of execution need additional features in Slurm.

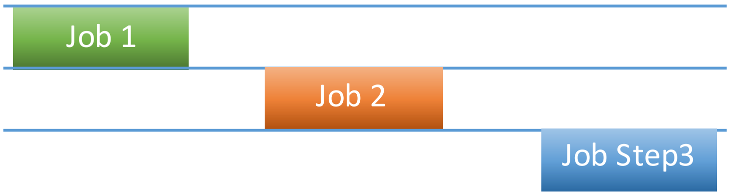


Figure 6: Typical workflow of dependent jobs supported by Slurm

Figure 6 shows a simple workflow already supported by Slurm. In this example each dependent sub-job is submitted as a separate individual job, together with all the dependency information. Slurm takes care of the dependencies between different jobs and makes sure that a job does not run unless all of its dependencies have been met.

Applications working together in terms of a workflow sometimes require intermediate data to be transferred from one job step to another. Usually, secondary memory like local disks or common network storage is used for this purpose by writing all the needed data into this memory at the end of the first job step and reading it again into the main memory at the start of the next one. Obviously, this approach for passing the intermediate data can be very time consuming.


Figure 7: Proposed workflow

To overcome such a problem, we propose a new feature in Slurm that guarantees some overlap of the execution of two sub-jobs having dependency. Figure 7 shows the execution of job steps in a job pack over time with overlaps. This overlap ensures that the two job steps may transfer the data between them while it resides in the main memory, hence avoiding the expensive secondary memory operations. This type of workflow also reduces the resource allocation as they are allocated to job steps only when they are required and making them available for the other jobs when not required. Such type of workflow is currently not supported by Slurm.

For the implementation of this proposed workflow, our initial idea is that the user provides the

estimated runtimes of each job step in a job pack. This information is provided as an amount of time for each of the job steps in a workflow for which their allocation should be deferred from the start time of the initial job step. These times are provided in a way to make sure that the execution times overlap. Slurm then ensures that the next job step in the workflow is allocated right after the given respective time has lapsed by reserving the time slots for each job step in advance.

We analysed the *begin* switch of Slurm job submission commands for our proposed workflow. A user may provide a begin time of a job being submitted and Slurm does not schedule the job for allocation until the time provided has lapsed. This switch is also available for each job step in a job pack. However, as mentioned earlier, all the job steps in a job pack are allocated with the requested resources together, hence Slurm puts all of the job steps on hold until the provided beginning times of all the job steps have lapsed. This clearly does not serve our requirements. We attempted to change the behaviour of begin switch for job packs but it proved to be very difficult due to the very complex nature of Slurm source code.

Therefore, we are introducing a new *delay* switch to provide the above mentioned functionality. The aim of providing a new switch to the Slurm commands instead of changing the behaviour of the existing one is to reduce the effort needed to change the Slurm source code. We are trying to be minimally invasive when changing the Slurm source code to support the above mentioned workflows. The implementation of delay switch is a work in progress.

Our current strategy is very simple and heavily depends on the estimations of execution times of the job steps provided by the users. An overestimation of the runtime may waste the resources as they will remain allocated for more time than actually needed. To overcome this problem, we propose developing a Slurm API in the future , which a job step can use to communicate with Slurm for early allocation of the next job step. If the resources required by the next job step are available, Slurm can allocate them earlier than reserved and hence data communication can be done between the two job steps. This process results in an early finish and deallocation of the earlier job step.

### 5.2.3 Efficient scheduling and management for shared global resources

In addition to the resources attached to the different compute nodes of the modules in the MSA, there are also shared global resources. There are currently two types of these resources planned for the prototype systems: Network Attached Memory (NAM) and Global Collective Engines (GCE). These shared resources will be attached to the EXTOLL network, and are subject to locality concerns given its torus-based topology.

Like other shared and limited resources, these will be managed to ensure fairness and system-wide efficiency metrics commonly optimized by scheduling heuristics. To manage them, the scheduler needs to be able to make future reservations. We have identified two ways to achieve reservations on these resources, by:

1. implementing reservation support in each manager, or
2. delegating their management to Slurm

We have decided for the later option because Slurm already has the required functionality to track resources and their reservations. The former would require the development of reserva-

tion support in both the NAM and GCE managers. Given reservation support, efficient schedules can be produced for jobs with NAM or GCE requirements.

Batch job definitions will be extended to include variables that represent these resources. The format will allow for the specification of any peculiarity of these resources. In the case of the NAM, it has been defined already that its allocations require the specification of the total memory and the number of access slots. The information required for GCE allocations is still to be defined.

Both GCE and NAM resources will be located at specific locations of the EXTOLL network. Their location in the network topology needs to be described in the resource metadata of Slurm. The scheduler can then make topology-aware allocations and reservations that maximize NAM, GCE and MPI performance for applications, taking into account node availability and other usual constraints.

# 6 System Monitoring and RAS Plane

In this section, we present the latest updates concerning the Data Centre Data Base (DCDB) tool, which is at the core of the task related to system monitoring and RAS plane. Specifically, in Section 6.1 we provide a brief recap of what DCDB is and what its main core functionalities and software components are; in Section 6.2 we provide an overview of the latest design decisions and enhancements, while in Section 6.3 we present a recently revised and improved high-level design specification of the tool.

## 6.1 Revisiting Context and State of the Art

Development of the DCDB monitoring tool [1] had already started within the DEEP project with a couple of specific requirements in mind:

i) 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.

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. In the original implementation of DCDB, Apache Cassandra [2] has been chosen as scalable and distributed key-value store.

In the DEEP-EST project, development of DCDB continues with the aim of maturing the code base, preparing it for deployment in a production environment, adding more data sources, and developing a feature rich visualisation and analysis frontend.

## 6.2 Recent Changes and Design Decisions

In this section, we provide an overview of the most recent architectural changes that had an impact to the core functionalities of DCDB.

### 6.2.1 User Interface Design Decisions

One of the key features for improving DCDB as a comprehensive system monitoring tool revolves around the development of an efficient way of accessing collected sensor data. Currently available options are via command-line interface tools (i.e., with the `dcdbquery` command) or via library (i.e., the `libdcdb` library). In addition to these mechanisms, we are currently implementing a more convenient solution for visualising data with a web Graphical User Interface (UI). The proposed GUI needs to be intuitive, illustrative and organised from a user/administrator perspective; for example, system administrators should be able to smoothly monitor live sensor data for evaluating the health of their monitored platforms, while end-users should be able to easily analyse and correlate different metrics associated with their executed applications. Multiple open-source dashboard tools exist from which DCDB could benefit in terms of data visualisation and analysis, such as Freeboard [3] (predominantly used for tracking KPIs in the IoT domain) or Dashing [4] (a popular open-source solution for e-commerce KPIs). After a thorough investigation, we concluded that the dashboard tool that best satisfies most of our requirements would be Grafana [5].

Grafana is an open-source visualisation tool which offers a general purpose dashboard, graph composers and runs as a web application. Grafana is well suited for efficiently representing time series data, supports a considerable number of storage back-ends and, to this date, benefits from a very large and active developer community. One key design feature of Grafana is its modular software architecture, in that contributors can improve and expand its functionalities through a convenient plug-in infrastructure. In this regard, all of the supported databases are integrated within Grafana via dedicated data source plug-ins, allowing for very rich query editors and tools for analysis. Unfortunately, Apache Cassandra, the key-value store upon which DCDB is built on, is not supported by Grafana and there is no official plan for considering its integration in the future. Nevertheless, we concluded that the benefits offered by Grafana in terms of data visualisation and analysis would greatly pay off the efforts necessary for implementing its integration with DCDB, even if that would potentially mean switching to another storage backend. Consequently, we conducted an analysis to evaluate which database option would eventually suit better our needs, taking into account the amount of work that a potential database change would mean in terms of additional software development work.

Among the many storage solutions that we analysed, two options seemed to be the most feasible, specifically *InfluxDB* and *OpenTSDB*. Table 1 sums up the advantages and disadvantages of each one of them, which we respectively illustrate as follows:

- InfluxDB [6] is an open-source time series database developed by InfluxData. It is written in Go and, similarly to Cassandra, is optimized for fast, high-availability storage and retrieval of time series data for operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics. InfluxDB has the tightest integration with Grafana and its data source plugin is built in the visualisation tool and shipped with it. It is very efficient in aggregating time series data and performs well with a relatively small resource footprint. On top of this, it further provides useful features in terms of retention policies and is relatively small and easy to configure and manage. Unfortunately, in contrast with tools like Cassandra, the open-source version of the database does not offer clustering, meaning that it is necessary to upgrade the software to the paid version for distributing the database across multiple servers.

- OpenTSDB [7] is a scalable, distributed time series database written in Java and built on top of HBase. In comparison with InfluxDB, its drawbacks outnumber the benefits, which can be easily summarised as (i) being a completely open-source solution and (ii) offering a considerably tight integration with Grafana through the OpenTSDB data source plugin (though not as sophisticated as the InfluxDB plugin). Differently from InfluxDB, OpenTSDB is not standalone and relies on HBase as its data storage layer. Additionally, it is primarily designed for generating dashboard visualizations, therefore it does not always return exact data for arbitrary time ranges. Finally, it is outperformed by both, InfluxDB and Cassandra, in terms of read/write performance and on-disk storage requirements, making it the less attractive solution of the two short-listed.

Under these premises, InfluxDB seemed to be the most promising candidate storage solution. However, we realised that the data distribution of Cassandra is an asset that we cannot exclude as it would be a desirable feature for systems based on a modular architecture such as the DEEP-EST platform, as well as for future Exascale prototypes. As DCDB is also distributed as open source software we concluded that all essential features of sub components should also be available as open source. Cassandra will then remain as the de-facto storage solution for DCDB and we will integrate it with Grafana by implementing a dedicated data source plugin.

### 6.2.2 Additional Minor Enhancements

DCDB has been further improved with the addition of two new features, namely sensor properties and sensor functions. In this section we briefly describe them as follows:

- *Sensor Properties*: each sensor is now associated with a dedicated entry in the database and a 64-bit mask, which specifies its properties. These last can be conveniently defined by the user via command line interface whenever it publishes a new sensor. Currently available properties are respectively:
  - INTEGRABLE: this flag specifies if the sensor data is quantifiable, making it eligible for mathematical operations.
  - MONOTONIC: this flag specifies if the sensor data follows a monotonic trend. This option is useful for triggering overflows of sensor data whose value resets after

| | Pros | Cons |
|---|---|---|
| InfluxDB | Tightest integration with Grafana<br>Efficient time-slot data aggregation<br>Easy to setup and maintain | Closed source clustering feature |
| OpenTSDB | Tight integration with Grafana<br>Completely open source | Worst read/write performance<br>Worst data compression |
| Apache Cassandra | Best read/write performance<br>Open source distributed solution<br>No need to rewrite DCDB core code | Lack of data source plug-in |

Table 1: Comparisons of different database options for Grafana integration

reaching a maximum allowed (e.g., for sensor data from energy meters).

The default value of the bit mask is 0, which corresponds to no sensor property specified.

- *Sensor Functions*: integrable sensors are eligible for the definition of sensor functions (and/or for mathematical expressions leading to the specification of virtual sensors). Sensor functions are applicable only to data belonging to a single sensor and can be conveniently defined by users within their queries via CLI tools. Currently available properties are:
  - **Delta**: the delta function simply returns the difference between two sensor values that are consecutive in time. For example, if the user queries delta(sensorName) within a specific interval of time $T$, the tool will return $sensorValue_t - sensorValue_{t-\Delta t}$ for all data included in the time frame $T$ (where $\Delta t$ is the interval of time between two consecutive timestamps).
  - **Derivative**: as the name suggests, the derivative function returns the derivative calculated between two sensor values that are consecutive in time. For example, if the user queries derivative(sensorName) within a specific interval of time $T$, the tool will return $(sensorValue_t - sensorValue_{t-\Delta t})/\Delta t$ for all data included in $T$ (where $\Delta t$ is the interval of time between two consecutive timestamps).
  - **Integral**: finally, similarly to the derivative, the integral function returns the integral calculated between two sensor values that are consecutive in time. For example, if the user queries integral(sensorName) within a specific interval of time $T$, the tool will return $(sensorValue_t - sensorValue_{t-\Delta t}) \cdot \Delta t$ for all data included in $T$ (where $\Delta t$ is the interval of time between two consecutive timestamps).

  Sensor functions will be further used as main building blocks for more sophisticated analytic tools.

## 6.3 High-level Design Specifications

The original DCDB design as developed and implemented within the DEEP project foresaw a push model for sensor data, i.e. 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. For example, there was a Pusher Daemon for Linux sysfs-based sensors, as well as pushers for the IPMI and SNMP protocols (for more details, please refer to Deliverable D5.1 [8]). An overview of the original software architecture of the tool is depicted in Figure 8. We have changed this design now in favour of a plug-in architecture in which a generic Pusher Daemon loads plug-ins during runtime which then provide the required connectivity to access actual sensors via specific protocols. The new architectural design of the tool is illustrated in Figure 9. For example, loading the sysfs plugin will allow for retrieving energy consumption data made available via Megware's EnergyMeters presented in Deliverable D4.1 [9]. Besides the Linux sysfs plugin, there are currently plugins available for IPMI, Linux perf events, and XML-formatted data sources. A plugin for SNMP is currently being ported from the previous stand-alone SNMP pusher. Additionally, a plugin for BACnet is currently being developed based on the open-source bacnet-stack [10]. An exhaustive list of

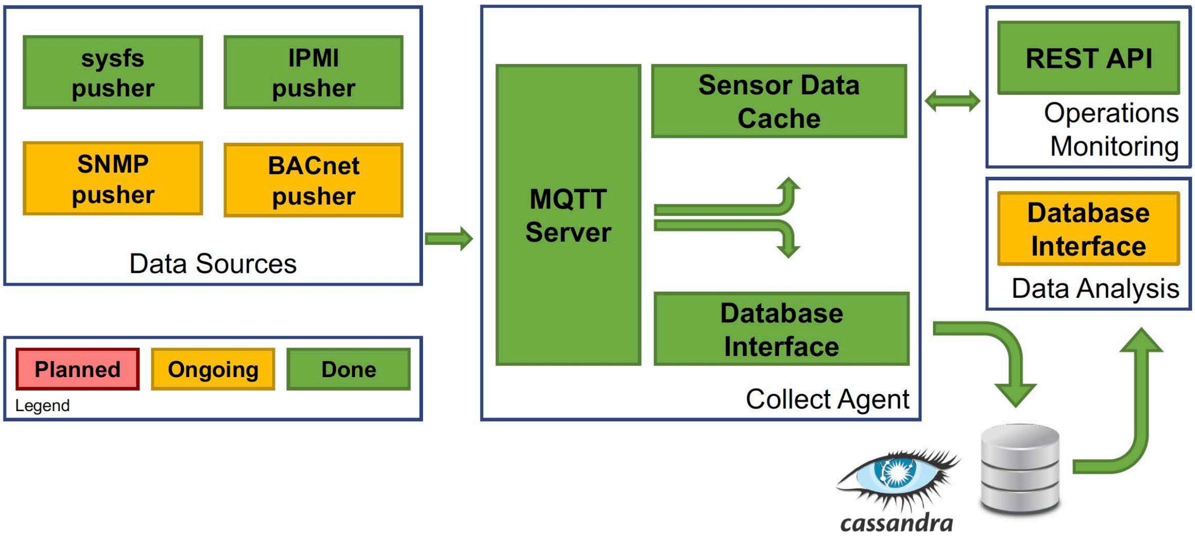


Figure 8: Previous software architecture of DCDB

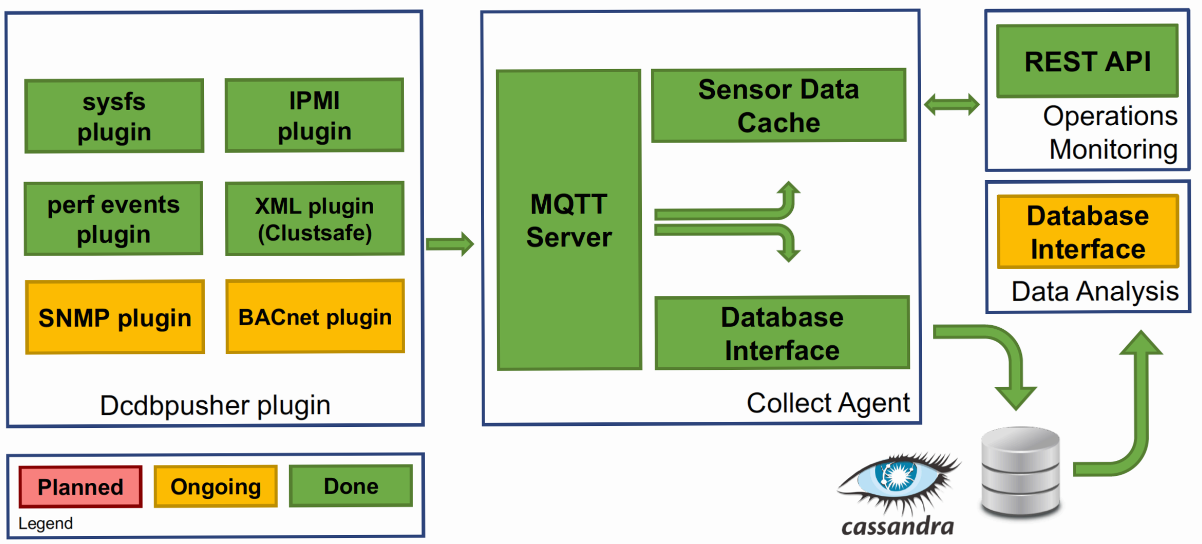


Figure 9: Current software architecture of DCDB

sensors that could be available for monitoring through DCDB will depend on the final shape of the hardware platform. Based on the collaborative interactions between hardware engineers in WP4 and software developers in WP5 and on the actual hardware design specifications, an indicative list of sensors that can be monitored with the associated pusher for gathering the relative data can be summarised as follows:

1. **For power and energy readings**:
   - per node/blade via Megware EnergyMeters (DC side) → *sysfs plugin*
   - per chassis via Megware Clustsafe (AC side) → *XML plugin*
2. **For in-rack water cooling readings**:
   - temperatures, flow rates, pressure, energy → *SNMP plugin*
3. **Performance Counters** → *perf events plugin*
4. **CPU Frequency** → *sysfs plugin*
5. **For infrastructure sensor data**:
   - flow rates, temperatures, etc. → *SNMP* or *BACnet plugin*

One of the main benefits of the new architecture is its modular design that minimizes code duplication as all protocol independent code is now encapsulated within the generic Pusher Daemon. Additionally, it allows for faster development of new plugins as the developer only needs to concentrate on the code gathering the sensor data - all functionality to push the data to the database is already in place.

From an administrator's point of view, the new architecture allows for easier deployment of Pusher Daemons. Previously, multiple Pusher Daemons for multiple protocols had to be deployed on a single machine and monitored. Under the new architecture, only a single daemon needs to be deployed. Additionally, an already running daemon can dynamically add new protocols at runtime without requiring a restart and potential loss of sensor information.

The new architecture will also facilitate the development of new features, such as dynamic re-configuration and a RESTful API to access sensor caches.

Most of the previously developed protocol-specific code could be easily ported to the new plug-in architecture and all previously available protocols are still available.

# 7 Summary

In this Deliverable, we have presented and discussed the design decisions taken since D5.1 that led us to the further work plan for WP5 also depicted here. We have revisited for each of the WP5 tasks the requirements as they were gathered and presented in D5.1, and subsequently derived an updated high-level description of the envisioned system software stack and its interfaces for the DEEP-EST prototype of a Modular Supercomputer Architecture (MSA).

The most important software development topics regarding interconnect management, network bridging, resource management, job scheduling and system monitoring are:

**A new interconnect management stack.** For the novel fabri$^3$ branch of the EXTOLL interconnect technology a likewise new management stack is currently under development. While being based on the well-known EXTOLL Management Program (EMP), the actual routing management can now be conducted directly by the EXTOLL hardware due to the integrated management CPU of fabri$^3$. Therefore, adaptations and extensions within the existing EMP system software stack become necessary and will be implemented for providing enhanced resiliency features.

**Update: GPGPU Support for EXTOLL.** For the new ESB design based on GPGPU accelerators, the EXTOLL driver stack will be extended to support high-bandwidth and low-latency access to the memory used by the code executed on the GPUs, by supporting the latest NVIDIA GPUDirect for RDMA API specification, as described in Section 2.4. Upper levels of the DEEP-EST SW stack, for example ParaStation MPI, will rely on this support to deliver high communication performance to the applications.

**An MPI bridge between InfiniBand and EXTOLL.** By now, the actual manifestation of the Network Federation has been determined, taking the following decisions concerning the different interconnect technologies: InfiniBand in the Cluster Module (CM), EXTOLL in the Extreme Scale Booster (ESB) and 40 GE Ethernet plus EXTOLL in the Data Analytics Module (DAM). Since ESB and DAM can directly communicate via EXTOLL, a single, daemon-based MPI bridge on dedicated gateway nodes between InfiniBand and EXTOLL is sufficient to forward MPI traffic between CM and DAM as well as between CM and ESB. In addition, this bridge will also be responsible for forwarding NAM requests from the CM to that part of the EXTOLL fabric the NAMs are attached to.

**Support for modularity, workflows and global resources.** Concerning resource management and job scheduling, recent developments in the Slurm upstream branch particularly accommodate the modularity of the DEEP-EST prototype. So, for instance, the integration of the Job Packs feature will especially help to run jobs across several MSA modules and to support workflows in modular architectures. In addition, the integration of Network Attached Memories (NAM) as globally shared resources into scheduling and process management is an important goal in DEEP-EST that will be accompanied by the realization of an efficient and modularity-aware job scheduling. Obviously, achieving this goal demands for a strong interlocking of Task 5.4 for Resource Management with Task 5.5 for Job Scheduling.

**A scalable and modular monitoring layer.** Last but not least, a scalable middleware for system monitoring is going to be developed for DEEP-EST. As inherited from the DEEP project, the Software Data Center Data Base (DCDB) monitoring solution will be further enhanced and

extended. For doing so, the existing DCDB code has already been revised comprehensively and restructured with respect to modularity and re-usability. In addition, the realization of an interface to the Grafana web application for visualizing the gathered sensor data is being tackled.

Prototype implementations of all these software components are either currently being kicked-off or are already under active development. The status of their integration, functionality and capability will then be presented in WP5's next Deliverable D5.3 in about half a year.

# 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 |
| **BAR** | Base Address Region: a memory region/address region exported by a PCIe device in the physical address space of the PCIe subsystem |
| **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** | short for Omni-Path Architecture (OPA), a communication architecture owned by Intel |
| **OPA** | see Omni-Path |
| **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-EST 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 |
| **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] *DCDB – DataCentre DataBase* [Online], Available: http://gitlab.lrz.de/dcdb

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

[3] *Freeboard: Dashboards For the Internet Of Things* [Online], Available: https://freeboard.io

[4] *Dashing: the Exceptionally Handsome Dashboard Framework* [Online], Available: http://dashing.io

[5] *Grafana: the Open Platform for Analytics and Monitoring* [Online], Available: https://grafana.com

[6] *InfluxDB, the TIme-Series Database* [Online], Available: https://www.influxdata.com/time-series-platform/influxdb

[7] *OpenTSDB: The Scalable Time Series Database* [Online], Available: http://opentsdb.net

[8] N. Eicker et al.: *DEEP-EST Deliverable 5.1: Collection of Software Requirements*, December 2017

[9] H. Cornelius and A. Auweter: *DEEP-EST Deliverable 4.1: Prototype Hardware Design*, June 2018

[10] *bacnet-stack* [Online], Available: http://bacnet.sourceforge.net

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

[13] 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

[14] Navarro A., Mateo S., Perez J.M., Beltran V., Ayguadé E. (2017) Adaptive and Architecture-Independent Task Granularity for Recursive Applications. In: de Supinski B., Olivier S., Terboven C., Chapman B., Müller M. (eds) Scaling OpenMP for Exascale Performance and Portability. IWOMP 2017. Lecture Notes in Computer Science, vol 10468. Springer, Cham Available: https://link.springer.com/chapter/10.1007/978-3-319-65578-9_12

[15] *Heterogeneous Resources and MPMD*, Presentation at the Slurm 2015 User Group Meeting [Online], Available: https://slurm.schedmd.com/SLUG15/Heterogeneous_Resources_and_MPMD.pdf

[16] *Heterogeneous Job Support in Slurm* [Online], Available: https://slurm.schedmd.com/

heterogeneous_jobs.html

[17] *Consumable Resources in Slurm* [Online], Available: https://slurm.schedmd.com/cons_res.html

[18] *srun man page* [Online], Available: https://slurm.schedmd.com/srun.html

[19] *sbatch man page* [Online], Available: https://slurm.schedmd.com/sbatch.html