SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

FP7-ICT-2011-7

DEEP

Dynamical Exascale Entry Platform
Grant Agreement Number: 287530

D4.7
Booster Resource Management with Static Allocation

Approved

Version: 2.0
Author(s): Suraj Prabhakaran (GRS), Sebastian Rinke (GRS), Michael Rauh (ParTec)
Date: 16.01.2015
**Project and Deliverable Information Sheet**

<table>
<thead>
<tr>
<th>DEEP Project</th>
<th>Project Ref. №: 287530</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Title:</strong></td>
<td>Dynamical Exascale Entry Platform</td>
</tr>
<tr>
<td><strong>Project Web Site:</strong></td>
<td><a href="http://www.deep-project.eu">http://www.deep-project.eu</a></td>
</tr>
<tr>
<td><strong>Deliverable ID:</strong></td>
<td>D4.7</td>
</tr>
<tr>
<td><strong>Deliverable Nature:</strong></td>
<td>Report</td>
</tr>
<tr>
<td><strong>Deliverable Level:</strong></td>
<td>PU *</td>
</tr>
<tr>
<td><strong>Contractual Date of Delivery:</strong></td>
<td>31 / March / 2014</td>
</tr>
<tr>
<td><strong>Actual Date of Delivery:</strong></td>
<td>27 / March / 2014</td>
</tr>
<tr>
<td><strong>EC Project Officer:</strong></td>
<td>Luis Carlos Busquets Pérez</td>
</tr>
</tbody>
</table>

* - The dissemination level are indicated as follows: PU – Public, PP – Restricted to other participants (including the Commission 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**

<table>
<thead>
<tr>
<th>Document</th>
<th>Title: Booster Resource Management with Static Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID:</strong></td>
<td>D4.7</td>
</tr>
<tr>
<td><strong>Version:</strong></td>
<td>2.0</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Approved</td>
</tr>
<tr>
<td><strong>Available at:</strong></td>
<td><a href="http://www.deep-project.eu">http://www.deep-project.eu</a></td>
</tr>
<tr>
<td><strong>Software Tool:</strong></td>
<td>Microsoft Word</td>
</tr>
<tr>
<td><strong>File(s):</strong></td>
<td>DEEP_D4.7_Booster_reource_management_static_allocation_v2.0_ECapprove</td>
</tr>
</tbody>
</table>

**Authorship**

- **Written by:** Suraj Prabhakaran (GRS), Sebastian Rinke (GRS), Michael Rauh (ParTec)
- **Contributors:** Felix Wolf (GRS)
- **Reviewed by:** Nils Meyer (UniReg), Estela Suarez (JUELICH)
- **Approved by:** BoP/PMT

**Document Status Sheet**

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Status</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>27.03.2014</td>
<td>Final</td>
<td>EC submission</td>
</tr>
<tr>
<td>2.0</td>
<td>16.01.2015</td>
<td>Final</td>
<td>EC approved</td>
</tr>
</tbody>
</table>

DEEP - 287530
Document Keywords

| Keywords:       | DEEP, HPC, Exascale, Resource Management, Scheduling |

Copyright notices

© 2011-2015 DEEP Consortium Partners. All rights reserved. This document is a project document of the DEEP project. All contents are reserved by default and may not be disclosed to third parties without the written consent of the DEEP partners, except as mandated by the European Commission contract 287530 for reviewing and dissemination purposes.

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

Project and Deliverable Information Sheet ................................................................. 2  
Document Control Sheet ................................................................................................. 2  
Document Status Sheet .................................................................................................. 2  
Document Keywords ...................................................................................................... 3  
Table of Contents .......................................................................................................... 3  
List of Figures ................................................................................................................ 4  
Executive Summary ....................................................................................................... 5  
1 Introduction ................................................................................................................. 5  
2 Overview of the Torque/Maui Batch System with the ParaStation Cluster Suite .......... 6  
3 Static Allocation Scenario .......................................................................................... 8  
  3.1 Booster Node Allocation .......................................................................................... 8  
  3.2 Offloading onto the Booster Nodes ......................................................................... 9  
4 Evaluation ................................................................................................................... 10  
5 Conclusion ................................................................................................................... 10  
6 References and Applicable Documents ...................................................................... 11  
7 List of Acronyms and Abbreviations ......................................................................... 12  

# List of Figures

Figure 1: Workflow of the Torque/Maui batch system with ParaStation Cluster Suite .......... 7  
Figure 2: Workflow of a static allocation scenario ........................................................... 9
Executive Summary

The DEEP System consists of two parts: an off-the-shelf HPC-cluster, or “Cluster” and a cluster of accelerators, or “Booster”. The Booster can be used to execute compute intensive parts of applications offloaded by Cluster Nodes (CNs). A primary goal of the DEEP platform is to provide transparency and flexibility in using the Booster Nodes (BNs). For this reason, the DEEP System strives to achieve BN-to-CN assignment in two ways:

- Static allocation – BNs are allocated before job start and released only when the job terminates.
- Dynamic allocation – BNs are allocated during runtime and can also be released at runtime.

Realising this requires an architecture-aware job management system with advanced resource management techniques to support the DEEP programming model. This document describes the requirements, design and implementation of the static allocation techniques for the DEEP system through the Torque/Maui batch system. For WP8, this document describes how to submit a job with BN requirements and the mechanism to offload computations onto the BNs. WP5 can use the described offloading mechanism as the basis for the final programming model based on OmpSs.

1 Introduction

Traditionally, Cluster Nodes (CN) are equipped with accelerators to which compute intensive kernels can be offloaded. To the present day off-the-shelf accelerator hardware cannot be used without a host CPU. A significant fraction of today’s high-performance CNs consist of (typically two) coherently coupled CPUs to which few accelerators are attached. The disadvantage of this approach is the tight coupling of the CPUs and the accelerators, which finally results in limited access to the accelerators and therefore induces scalability issues. The DEEP System exhibits a novel accelerator-cluster architecture that is different from existing approaches. To overcome typical limitations, the DEEP project develops a separate cluster of network-attached accelerators called Booster Nodes (BN). The BNs can be assigned to the CNs based on application needs. The system offers a programming model based on the dynamic process management facilities in MPI to offload kernels onto the BNs. In particular, the MPI_Comm_spawn() mechanism enables spawning parallel kernels onto the allocated accelerators. Furthermore, the architecture intends to support a flexible usage of BNs where applications can adjust the number of BNs assigned to them at runtime. For example, extra BNs can be allocated to CNs when a computational phase demands more resources. Similarly, when certain BNs are not required anymore, they can be released at runtime so that they can be used by other applications.

This approach requires an architecture-aware batch job management system that closely cooperates with the programming model to support the static and dynamic Booster Node allocation and a transparent computation offloading scheme. Existing batch systems lack the required support for such network-attached accelerator architectures and hence cannot be inherently used. This deliverable focuses on the static allocation scheme for the DEEP architecture and presents the requirements, design and implementation for the same along with considerations towards the dynamic allocation scheme.

The DEEP System will be managed by the Torque/Maui batch job management system. The Torque/Maui ensemble is well-known for advanced scheduling and can tightly integrate with the ParaStation process management (psmgmt) through the ParaStation MOM (psmom). This document provides an overview of the Torque/Maui batch system integrated with the
ParaStation Cluster Suite [3] and explains the necessary extensions made to them to enable the static allocation facility.

2 Overview of the Torque/Maui Batch System with the ParaStation Cluster Suite

The Torque/Maui batch system is one of the most commonly used middleware for batch job control. The Torque resource manager [4] is based on the PBS project [5] extended to improve scalability and fault tolerance and is currently maintained by Adaptive Computing. Torque is usually integrated with sophisticated schedulers such as Maui [12], which provides advanced scheduling features such as job prioritization, fairshare and backfill scheduling.

A Torque/Maui cluster consists of a headnode, a frontend, and many compute nodes. The headnode runs the pbs_server daemon (server) and the Maui scheduler daemon. The compute nodes run the pbs_mom daemon (mom). A number of client commands to communicate with the server are provided to users for tasks such as job submission, alteration and checking the status of a job. They are installed on the frontend. The client submits a job through the qsub command by specifying the number of nodes, the number of processors per node, the duration for which resources are required (walltime of the job), and other software or hardware requirements. The job is then queued at the server. When resources are allocated for this job by the Maui scheduler, the server sends the job to one of the nodes allocated for this job (called mother superior) and updates the state of this job in the queue as running. The mother-superior then starts execution of the job.

The Maui scheduler communicates with the server and schedules jobs iteratively. A scheduling iteration is followed by a period of sleeping or processing of external commands. Maui will instantly start a new iteration when (i) a job or resource state change occurs, (ii) a reservation boundary event occurs, (iii) an external command to resume scheduling is issued or (iv) a configurable timer expires. During each iteration, Maui obtains the most recent information about resources and jobs from Torque and updates the historical statistics and usage information of all the jobs. Then, jobs meeting a minimum scheduling criterion, based on throttling policies and job states, are selected and considered for scheduling. The selected jobs are prioritized according to various policies and scheduled in the order of their priorities. When a lack of resources prevents the idle job with the highest priority from starting, the earliest time when the resources are available for this job is determined and a reservation is created. Maui continues to create reservations for $N$ such highest priority jobs where $N$ can be configured using the ReservationDepth parameter. Jobs that are not reserved are then backfilled out of order. Backfilling is a strategy of increasing resource utilization by running low priority jobs out of order as long as they do not disturb the high priority reservations. A higher ReservationDepth leads to a more conservative backfilling while a lower ReservationDepth allows more jobs to get backfilled.

A Torque/Maui cluster integrated with the ParaStation middleware uses a ParaStation process management daemon (psid) instead of the pbs_mom in the nodes. The ParaStation MOM (psmom) provides all the functionalities of the pbs_mom and is implemented as a plugin to the psid. Using the psmom plugin, the psid is therefore the only daemon needed on the compute nodes for batch system operation. Furthermore the ParaStation MOM offers a fast and reliable job startup and offers various unique features such as:

- Advanced accounting for all processes of serial and parallel jobs
- Secure communication between the ParaStation process management and the Torque batch system
- Reliable and highly scalable node-to-node communication
- On-the-fly reconfiguration and information gathering
An example of a typical workflow of the batch system is illustrated in Figure 1. The client submits a job through the qsub command as shown below.

```
qsub -l nodes=4 jobscript.sh
```

The new job request is queued at the server triggering a scheduling iteration in the Maui scheduler. Once resources are allocated, the server receives a nodelist (or hostlist) with the list of nodes allocated for this job from the Maui scheduler. The server selects the first node in the nodelist as the mother-superior node and sends the entire job information to this node. The jobscript is then executed in the mother-superior node through the psmom plugin.

Parallel jobs using the ParaStation MPI start execution through the ParaStation mpiexec command, which internally performs the following two steps. First, a new partition in the ParaStation process management is created using the resources in the nodelist. The size of the partition will be implicitly set to the mpiexec option “--np” (the number of processes to start). It can also be explicitly set by using the mpiexec option “--usize” or by setting the `MPIEXEC_UNIVERSE_SIZE` environment variable. Second, the executable specified will be spawned onto the newly created partition. An example mpiexec command with “--usize” explicitly set is shown below.

```
mpiexec --usize 4 --np 4 ./application
```

After application termination, the psmom of the mother-superior node informs the server of the completion of the job. The server updates the state of the nodes in its internal database and the nodes are free to be used by other jobs.
3 Static Allocation Scenario

In the static allocation scenario, the BNs are assigned to CNs before job start and remain allocated until job termination. The extensions made to the batch system and their usage is detailed in the section.

3.1 Booster Node Allocation

The batch system is configured to be aware of different node types in the DEEP System through the hardware-type property which can be set for each node’s hostname in the Torque server configuration file. We distinguish between the cluster and the booster node type. An example is depicted below.

```
host1    cluster
host2    cluster
host3    booster
host4    booster
```

At job submission, users can use these node types to specify their resource requirements with the `qsub` command. The following shows an example.

```
qsub -l nodes=2:cluster+2:booster jobscript.sh
```

The above command requests two CNs and two BNs for the job. For the user’s convenience, we extended the `qsub` command to support the `bpm` (boosters per node) keyword. It allows to indicate the number of booster nodes required per compute node by a job.

```
qsub -l nodes=2:bpm=1 jobscript.sh
```

The above command requests two CNs with one BN per CN (i.e., two BNs in total). The two examples above are only ways of indicating resource requirements and do not prevent a CN from using any BNs allocated for its job. It may be more convenient for users to specify resource requirements with the `bpm` keyword when their applications tend to use a fixed number of BNs per CN. Users with applications that have a more irregular usage of BNs per CN may prefer the first method.

When requesting different node types, it is not guaranteed that the nodelist will be populated with the CNs first followed by the BNs. However, this ordering is required to ensure the correct default behavior of starting the application on the CNs. Therefore, an extension to the Torque server was required to ensure a correctly ordered nodelist with the CNs followed by the BNs. This also ensures that the mother-superior selected by the server is always a CN and not a BN.
3.2 Offloading onto the Booster Nodes

For the static resource allocation scenario the user must be able to choose on which type of node new processes will be started. This can be done through the mpiexec in the job script or using the MPI_Comm_spawn() from within the application executing on the CN. To support this functionality, a new mpiexec option “--nodetype” was introduced in the ParaStation MPI. Consequently, the protocol for requesting a partition was extended to handle the new node type information and the partition management code was extended to support the allocation of arbitrary nodes. An example of an mpiexec command starting the application on two CNs is shown below.

```bash
mpiexec --usize 4 --np 2 --nodetype “cluster” ./application
```

Note that in the absence of the nodetype parameter, by default, all processes will be started in the exact order of the nodelist the batch system provides. When calling MPI_Comm_spawn() for starting new processes, the partition which was created with the mpiexec call is used to determine if enough resources are left to satisfy the spawn request. Therefore, it is important to explicitly set the partition size to the total number of nodes allocated for this job by the batch system. Similar to mpiexec, to spawn processes on the BNs, the MPI_Info argument of the MPI_Comm_spawn() provides a new “nodetype” key which can be set to the value “booster”. The code below shows an example of spawning two processes on Booster Nodes. An illustration of the scenario is presented in Figure 2.

```c
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, “nodetype”, “booster”);
MPI_Comm_spawn(“child”,
    MPI_ARGV_NULL,
    2,
    info,
    0,
    MPI_COMM_WORLD,
    &intercomm,
    MPI_ERRCODES_IGNORE);
```

Figure 2: Workflow of a static allocation scenario
When users want to directly execute their application on the BNs, the nodetype of the mpiexec command can be set to “booster” as shown in the example below.

```bash
mpiexec --np 2 --nodetype "booster" ./application
```

### 4 Evaluation

The above mentioned functionalities tightly integrate with the DEEP programming model and provide a convenient means for managing BNs both to the users as well as to the MPI. The resource allocation in Torque/Maui does not incur any additional overhead by requesting different node types. Therefore, the response time (defined as the time taken after job submission until starting the first process) for a job depends only on the availability of resources, workload and priority of the job.

Similarly, by specifying node types, partition creation and partition management do not produce any additional overhead in starting the application or in spawning processes on the BNs, as compared to the same operation in homogeneous architectures. With respect to MPI_Comm_spawn(), its duration is dominated by the process spawning through the ParaStation MOM and the protocol for communicator creation rather than the node selection in the ParaStation partition management.

### 5 Conclusion

The special DEEP architecture also requires a special batch system that can support the programming model which employs MPI-2 dynamic process management facilities. The extensions to Torque and the ParaStation Cluster Suite were made to enable the static allocation scenario. The static allocation scheme offers a flexible usage of allocated BNs according to the needs of the application.

The DEEP System also intends to offer sophisticated Booster management schemes through runtime allocation and deallocation of Booster Nodes to a job. One of the first requirements for such a dynamic allocation is a job management system which can distinguish between different node types. Also, the partition creation and management based on the node type is essential. This subset of prerequisites for the dynamic allocation is common to the static allocation implemented for this deliverable. That is, our work described here will be the foundation for developing the dynamic allocation facilities in the next deliverable, D4.8, due in month 31.
6 References and Applicable Documents

7 List of Acronyms and Abbreviations

**B**
- BN: Booster Node (functional entity)

**C**
- CN: Cluster Node (functional entity)

**D**
- DEEP: Dynamical Exascale Entry Platform: EU-FP7 Exascale Project led by Forschungszentrum Juelich
- DEEP Architecture: Functional architecture of DEEP (e.g. concept of an integrated Cluster Booster Architecture)
- DEEP System: The production machine based on the DEEP Architecture developed and installed by the DEEP project

**G**
- GRS: German Research School for Simulation Sciences GmbH, Aachen, Germany

**M**
- MPI: Message Passing Interface: API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages
- MOM: The ParaStation MOM (psmom) which is a plugin to the ParaStation process management daemon (psid) responsible for communicating with the central batch system server and executing applications on compute nodes

**P**
- ParaStation Cluster Suite: The Parastation MPI consisting of the MPI library (psmpi), the communication library (pscom) and the process management library (psmgmt)
- ParTec: ParTec Cluster Competence Center GmbH, Munich, Germany
- PBS: Portable Batch System – the original implementation from which Torque resource manager is cloned
- Psid: The ParaStation MPI process management daemon executed on each node of the DEEP system
- Psmgmt: ParaStation MPI process management library
- Psmom: ParaStation MOM is a plugin to the ParaStation process management daemon (psid) responsible for communicating with the central batch system server and executing applications on the nodes

**W**
- WP: Work Package