Project Number 611411

D1.2 – Dynamic Resource Allocation Requirements

Version 1.0
30 April 2014
Final

EC Distribution

University of York, University of Stuttgart

Project Partners: Aicas, Bosch, CNRS, Rheon Media, The Open Group, University of Stuttgart, University of York

Every effort has been made to ensure that all statements and information contained herein are accurate, however the Project Partners accept no liability for any error or omission in the same.

© 2014 Copyright in this document remains vested in the DreamCloud Project Partners.
# Project Partner Contact Information

<table>
<thead>
<tr>
<th>Aicas</th>
<th>Bosch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fridtjf Siebert</td>
<td>Jochen Härdtlein</td>
</tr>
<tr>
<td>Haid-und-Neue Strasse 18</td>
<td>Robert-Bosch-Strasse 2</td>
</tr>
<tr>
<td>76131 Karlsruhe</td>
<td>71701 Schwieberdingen</td>
</tr>
<tr>
<td>Germany</td>
<td>Germany</td>
</tr>
<tr>
<td>Tel: +49 721 663 96823</td>
<td>Tel: +49 711 811 24517</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:siebert@aicas.com">siebert@aicas.com</a></td>
<td>E-mail: <a href="mailto:jochen.haerdtlein@de.bosch.com">jochen.haerdtlein@de.bosch.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CNRS</th>
<th>Rheon Media</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gilles Sassatelli</td>
<td>Raj Patel</td>
</tr>
<tr>
<td>Rue Ada 161</td>
<td>20 Leighton Avenue</td>
</tr>
<tr>
<td>34392 Montpellier</td>
<td>Pinner Middlesex HA5 3BW</td>
</tr>
<tr>
<td>France</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>Tel: +33 4 674 18690</td>
<td>Tel: +44 7547 162920</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:sassatelli@lirmm.fr">sassatelli@lirmm.fr</a></td>
<td>E-mail: <a href="mailto:raj@rheonmedia.com">raj@rheonmedia.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The Open Group</th>
<th>University of Stuttgart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scott Hansen</td>
<td>Bastian Koller</td>
</tr>
<tr>
<td>Avenue du Parc de Woluwe 56</td>
<td>Nobelstrasse 19</td>
</tr>
<tr>
<td>1160 Brussels, Belgium</td>
<td>70569 Stuttgart</td>
</tr>
<tr>
<td>Tel: +32 2 675 1136</td>
<td>Germany</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:s.hansen@opengroup.org">s.hansen@opengroup.org</a></td>
<td>Tel: +49 711 68565891</td>
</tr>
<tr>
<td></td>
<td>E-mail: <a href="mailto:koller@hlrs.de">koller@hlrs.de</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>University of York</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Leandro Soares Indrusiak</td>
<td></td>
</tr>
<tr>
<td>Deramore Lane</td>
<td></td>
</tr>
<tr>
<td>York YO10 5GH</td>
<td></td>
</tr>
<tr>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td>Tel: +44 1904 325571</td>
<td></td>
</tr>
<tr>
<td>E-mail: <a href="mailto:leandro.indrusiak@cs.york.ac.uk">leandro.indrusiak@cs.york.ac.uk</a></td>
<td></td>
</tr>
</tbody>
</table>
## DOCUMENT CONTROL

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>First overview of dynamic resource allocation requirements.</td>
<td>7 March 2014</td>
</tr>
<tr>
<td>0.3</td>
<td>Requirements list revision due to the project partners' comments</td>
<td>10 April 2014</td>
</tr>
<tr>
<td>0.5</td>
<td>Added requirements for HPC platforms</td>
<td>15 April 2014</td>
</tr>
<tr>
<td>0.7</td>
<td>Further updates to requirements list descriptions</td>
<td>16 April 2014</td>
</tr>
<tr>
<td>0.9</td>
<td>Minor revision due to the industrial project partners' comments</td>
<td>26 April 2014</td>
</tr>
<tr>
<td>1.0</td>
<td>Minor QA revisions to footers and styles</td>
<td>30 April 2014</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

1. Introduction ............................................................................................................................................... 1
   1.1 Achievements Indicator for Dynamic Resource Allocation ................................................................. 3
   1.2 Prioritisation of requirements .................................................................................................................. 3
   1.3 Structure of this document ....................................................................................................................... 4

2. State of The Art in Dynamic Resource Allocation .................................................................................. 5

3. Dynamic Resource Allocation Requirements in DreamCloud .................................................................. 8
   3.1 Introduction .............................................................................................................................................. 8
   3.2 Variety of Relevant Dynamic Resource Allocation Heuristics ............................................................... 8
      3.2.1 Control-theoretic-based heuristics .................................................................................................... 8
      3.2.2 Evolutionary Algorithms ............................................................................................................... 9
      3.2.3 Market-inspired heuristics ............................................................................................................ 9
      3.2.4 Swarm-Intelligence inspired heuristics .......................................................................................... 9
   3.3 Requirements List ................................................................................................................................... 10
      3.3.1 Objectives of dynamic resource management should be configurable ........................................... 10
      3.3.2 Specified hard real-time constraints shall not be violated .............................................................. 10
      3.3.3 Dynamic resource allocation shall be used to provide different levels of performance guarantees .... 10
      3.3.4 The average latency of jobs shall be minimised ............................................................................ 10
      3.3.5 The total energy dissipation of jobs shall be minimised ................................................................. 11
      3.3.6 Communication overhead parameters shall be predictable ............................................................ 11
      3.3.7 Dynamic resource allocation overhead shall be predictable and bounded .................................. 11
      3.3.8 The dynamic resource allocation mechanisms shall cope with dynamic workload ................... 11
      3.3.9 The dynamic resource allocation mechanisms shall not limit hardware scaling ....................... 11
      3.3.10 The dynamic resource allocation mechanisms shall cope with limited information about the state of the overall system ........................................................................................................................................ 12
      3.3.11 The dynamic resource allocation mechanisms shall respect mapping constraints that restrict the allowed computational unit ........................................................................................................................................ 12
      3.3.12 The dynamic resource allocation mechanisms shall consider cost, runtime and power efficiency for different type of resources available to a multi-typed job .................................................................................................................. 12

4. Operating system and Virtual machine support .......................................................................................... 15
   4.1 Introduction ............................................................................................................................................ 15
   4.2 Relation with Task Scheduling ............................................................................................................. 15
   4.3 Interface of Relevant Dynamic Resource Allocation Heuristics ........................................................... 16

References ....................................................................................................................................................... 19
EXECUTIVE SUMMARY

This deliverable describes the dynamic resource allocation requirements for heuristic algorithms to be developed within the DreamCloud project. It encompasses the list of the requirements, together with their explanation and justification for three platforms used in the project: Embedded Clouds, Micro Clouds, and High Performance Clouds. Due to inherent differences between these platforms and the variety of potential applications, a bunch of assorted resource allocation algorithms has to be proposed. These algorithms are planned to stem from such diverse domains as control theory, evolution, market, and swarm intelligence. The most crucial requirements for these algorithms are enlisted and briefly explained together with a presentation of the coverage of these requirements by the introduced heuristic classes. This information can be used as a guideline during selecting a proper technique for a particular application.

The majority of dynamic resource allocation heuristics that are planned to be used in the DreamCloud project are based on the extraction of information about applications and the target platform from internal representations, which is described in Deliverable 1.3. It is also assumed that the underlying operating system and virtual machine is capable of providing the necessary performance metrics in real-time. The most important of these metrics are enumerated in this document. Also, the requirements towards a task scheduler used by a target operating system or virtual machine are described.
1. INTRODUCTION

The aim of the DreamCloud project is to add the capability of dynamic resource allocation to contemporary and future computer systems. Since many-core systems are of mounting popularity in the majority of domains and applications, from embedded clouds up to supercomputing centres, the typical system is likely to be dynamic, with a high density of computing and communication resources. In order to utilise such systems with adequate performance, it is essential that resource allocation algorithms should make decisions frequently enough to follow the system dynamics. At each decision point, such algorithm shall consider dozens or hundreds of potential allocation possibilities, including selection of a processor to execute each application task, communication links to be used to exchange data between tasks, etc.

Taking into account the sheer size of the allocation space together with additional non-functional requirements imposed to the resource management subsystem (time, energy), it is practically impossible to make an optimal allocation decision. This is the reason why the DreamCloud project takes a heuristic approach to the allocation decision making. A number of various heuristics are planned to be devised, each of them answering different subset of system runtime requirements. Together with the heuristics, their description of capabilities and applications will be provided. The most important trade-offs will be explored within the project, namely how to dynamically balance the quality of the resource allocation and the overhead of obtaining it, according to the constraints on each kind of system regarding performance predictability. These heuristics will be implemented and tested using popular benchmarks, and also proprietary benchmarks provided by the industrial project partners, on various platforms. Then it would be simpler to choose an appropriate allocation technique for a given application and platform, and to forecast the final outcome.

The dynamic resource allocation algorithms used in the DreamCloud project require information about applications to be executed and the target platform. Both these information are provided by means of internal representations (respectively, the internal application model and internal platform model shown in Figure 1), see also D.1.3. These internal representations are intended to enable heuristics to predict performance and analyse the impact of various allocation decisions.

DreamCloud will use a hierarchical composition of allocation heuristics to propagate performance and energy guarantees bottom-up from individual cores to large-scale platforms. Heuristics operating at the lower levels will provide strict guarantees for time-critical applications and will locally minimise their energy dissipation while avoiding thermal imbalances. They will also dynamically profile spare capacity that can be offered to higher level allocation heuristics with less strict guarantees. Those higher level heuristics, in turn, will aim for minimisation of costly communications between many-core processors (which will be orders of magnitude higher than using the many-core interconnect), manage heterogeneity, balance load and maximise value.
Figure 1: Dynamic resource allocation heuristics and their internal representations of application load and platform resources

Low level allocation heuristics operating at the many-core level will have to guarantee hard real-time constraints to critical application tasks. This will only be possible for applications that have been profiled a priori so their execution and communication patterns can be accurately represented by an application model. Such applications will not be highly dynamic, and will exhibit modal behaviour, so that distinct modes of operation can be analysed at design time, and dynamic allocation will be based on pre-defined alternatives (thus the number of allocation decisions during runtime is minimal).

Low level allocation heuristics must also guarantee soft real-time constraints to non-critical but performance-sensitive application tasks. Such tasks may be known a priori, and they can be dealt with as described in the item above and follow pre-defined allocation alternatives. If they are to be allocated on demand (e.g., an application that has been received from a heuristic operating at a higher level), a different approach must be developed.

Low level heuristics must also profile available capacity and make it available to allocation heuristics operating at higher levels. This project will investigate two approaches. One of them is a side effect of the algebraic approach proposed to evaluate allocations. The results of the operations over application models based on time intervals will not only show delays on intervals sharing resources, but will also result on similar patterns of idleness on each resource (which are denoted as different dimensions over the algebraic space). Such idleness patterns can be used as indication of free capacity. While the first approach is an open-loop analysis, it can lose accuracy if the actual behaviour of the application deviates from the computation and communication patterns represented in the application model (which is bound to happen in highly variable applications). To increase the accuracy of the capacity profiling, a closed-loop approach will use available monitoring infrastructure to update its internal
representation of the application (e.g. if tasks or communications underrun or overrun their budgeted allocation, if overheads are larger or smaller than predicted).

Deliverable 1.2 focuses on DreamCloud’s requirements for dynamic resource allocation heuristics. The findings described here will directly contribute to the technical objective TO2 from the DreamCloud’s Description of Work, namely the “Creation of novel resource allocation heuristics that are sufficiently lightweight to be applied during runtime, and that are able to take into account timing guarantees expressed within application models”.

1.1 ACHIEVEMENTS INDICATOR FOR DYNAMIC RESOURCE ALLOCATION

Dynamic Resource Allocation metrics will be based upon the degree to which a system’s runtime resource requirements are optimised (e.g. less energy, higher utilisation). Allocation techniques should:

- be applied at runtime with timing and energy overheads that should not exceed 1% of the time and energy costs of the application workload they are allocating;
- achieve at least 20% higher resource utilisation than a static allocation that provides the same timing guarantees;
- achieve at least 30% less energy dissipation than a static allocation that provides the same timing guarantees;
- for time-critical applications, enable timing guarantees to at least 80% of the cases that could be allocated statically according to at least one of all the possible mapping decisions (if computable in reasonable time), but with a reduction of the runtime overhead caused by a dynamic allocation of at least three orders of magnitude;
- enable thermal imbalance management that is at least 20% better than statically allocated solutions.

In some particular situations, for example in embedded applications, these numbers may not be achievable since the application itself might not have such a high potential as there might not be so much difference in the load over runtime.

1.2 PRIORITISATION OF REQUIREMENTS

Throughout this deliverable priorities for requirements are identified in terms of the modalities SHALL, SHOULD and MAY, which are defined as follows:

- **SHALL** is used to denote an essential requirement. A typical target system could not be used, would not work, or cannot be validated if this requirement is not fulfilled. SHALL requirements are of highest priority for validation of the DreamCloud technologies.
- **SHOULD** is used to denote a requirement that would help a typical system be easier to use, or to work better, even if it is not essential; in that case a trade-off
can be achieved between development costs on the technology side and user benefit on the system side.

- **MAY** is used to denote a requirement that can lead to a benefit in order to fulfil an additional evaluation criterion or increase the usefulness of the technology. The fulfilment of the requirement is interesting but only in view of available resources and research and development partner interests.

During the evaluation tasks later in the project the industrial partners will verify the degree to which each of the requirements at each priority level has been fulfilled by the produced application models.

### 1.3 STRUCTURE OF THIS DOCUMENT

This deliverable is structured as follows:

- Section 2 briefly reviews the state of the art in dynamic resource allocation.

- Section 3 describes the general requirements for dynamic resource allocation and provides a consolidated and sorted listing of all the application dynamic resource allocation requirements, and their mappings to the research and development workpackage(s) where they will be addressed.

- Section 4 describes the requirements on the underlying virtual machine and operating system in order to make it possible to apply the dynamic resource heuristics developed in the project.

Sources for additional information are referenced or footnoted throughout the document.
2. **State of the Art in Dynamic Resource Allocation**

Resource allocation is one of the most complex problems in large many-core and distributed systems, and in general it is considered NP-hard [5]. The theoretical evidence shows that the number of possible allocations of application tasks grows factorially with the increase of the number of processing cores. The empirical evidence points in the same direction, in that for a realistic many-core embedded system (40-60 application components, 15-30 processing cores) a well-tuned search algorithm had to statically evaluate hundreds of thousands of distinct allocations before it finds one solution that meets the system’s performance requirements.

This problem was first addressed from the cluster/grid computing point of view, but more recently the fine-grained allocation of tasks within many-core processors has also received significant attention due to its critical impact on performance and energy dissipation. In the following subsections, we consider allocation mechanisms at both grid and many-core level, and review the most significant trends and achievements in terms of guaranteed performance and energy efficiency.

**Allocation Techniques for Guaranteed Performance**

There are numerous multiprocessor scheduling and allocation techniques that are able to meet real-time constraints, each of them under a different set of assumptions. In [11], a very comprehensive survey has been conducted covering techniques that can be applied both at the grid or many-core level, but all of them assume that the platform is homogeneous and tasks are independent (i.e. do not explicitly consider communication costs). Many of them also assume that the allocation is done statically, or do not take into account the overheads of dynamically allocating and migrating tasks (i.e. context saving and transferring). Heterogeneous platforms are considered but communication costs and overheads are still not taken into account [26].

Significant research on resource reservation has been done, aiming to increase time-predictability of workflow execution over HPC platforms [25]. Many approaches use a priori workflow profiling and use estimation of task execution times and communication volumes to plan ahead which resources will be needed when tasks become ready to execute. Just like in static allocation, resource reservation policies significantly reduce the utilisation of HPC platforms. A reduction of 20-40% in the utilisation is not unusual [38].

Allocation and scheduling heuristics based on feedback control have been used in HPC systems, aiming to improve platform utilisation without sacrificing performance constraints [12][21]. Most cases concentrate on controlling the admission and allocation of tasks over the platform based on a closed-loop approach that monitors utilisation of the platform as well as performance metrics such as task response times [13].

Many cloud-based and grid-based HPC systems use allocation and scheduling heuristics that take into account not only the timing constraints of the tasks but also their value (economic or otherwise). This problem has been well-studied under the model of Deadline and Budget Constraints (DBC) [6], where each task or taskflow has a fixed deadline and a fixed budget. State-of-the-art allocation and scheduling techniques target
objectives such as maximising the number of tasks completed within deadline and/or budget [36], maximising profit for platform provider [18] or minimising cost to users [33] while still ensuring deadlines. Several approaches to the DBC problem use market-inspired techniques to balance the rewards between platform providers and users [40]. A comprehensive survey that has been conducted reviews several market-based allocation techniques supporting homogeneous or heterogeneous platforms, some of them supporting applications with dependent tasks modelled as DAGs [42].

At the many-core level, there are a few allocation techniques that take into account both the computation and communication performance guarantees. Such techniques are tailored for specific platforms e.g. many-cores based on Network-on-Chip (NoC). To guarantee timeliness, all state-of-the-art approaches rely on a static allocation of tasks and communication flows. A multi-criteria genetic algorithm has been used to evolve task allocation templates over a NoC-based many-core aiming to reduce their average communication latency [3]. A further approach also used a genetic algorithm that could find an allocation that can meet hard real-time guarantees on end-to-end latency of sporadic tasks and communication flows over many-cores that use priority-preemptive arbitration [28]. Stuijk [35] proposed a constructive heuristic to do static allocation of synchronous dataflow (SDF) application models, which constraint all tasks to read and write the same number of data tokens every time they execute. The allocation guarantees the timeliness of the application if the platform provides fixed-latency point-to-point connection between processing units. The same author relaxes some of the assumptions of SDF applications (i.e. allows for changes on token production and consumption rates during runtime) and proposes analytical methods to evaluate worst-case throughput and to find upper bounds for buffering for a given static allocation.

**Allocation Techniques for Energy-Efficiency**

Most allocation techniques addressing energy efficiency operate at the many-core processor level, mainly because of the difficulties of dealing with energy-related metrics at larger system granularities.

Hu et al. [16] and Marcon et al. [24] estimate the energy consumption according to the volume of data exchanged by different application tasks over the interconnection network. Such approaches lack in accuracy as they do not take into account runtime effects such as network congestion or time-varying workloads. Thus, research approaches on energy-aware dynamic allocation techniques have been proposed.

An iterative hierarchical dynamic mapping approach has also been used to reduce energy consumption of the system while providing the required QoS [32]. In such strategy, tasks are firstly grouped by assigning them to a system resource type (e.g. FPGA, DSP, ARM), according to performance constraints. Then, each task within a group is mapped, minimising the distance among them and reducing communication cost. Finally, the resulting mapping is checked, and if it does not meet the application requirements, a new iteration is required.

Chou and Marculescu [8] introduce an incremental dynamic mapping process approach, where processors connected to the NoC have multiple voltage levels, while the network has its own voltage and frequency domain. A global manager (OS-controlled
mechanism) is responsible for finding a contiguous area to map an application, and for defining the position of the tasks within this area, as well. According to the authors, the strategy avoids the fragmentation of the system and aims to minimize communication energy consumption, which is calculated according to Ye et al. [41]. This work was extended in [9][10] to incorporate the user behaviour information in the mapping process. The user behaviour corresponds to the application profile data, including the application periodicity in the system and data volume transferred among tasks. For real applications considering the user behaviour information, the approach achieved around 60% energy savings compared to a random allocation scenario.

Holzenspies et al. [14] investigate a run-time spatial mapping technique with real-time requirements, considering streaming applications mapped onto heterogeneous MPSoCs. In the proposed work, the application remapping is determined according to information that is collected at design time (i.e. latency/throughput), aiming to satisfy the QoS requirements, as well as to optimize the resources usage and to minimise the energy consumption. A similar approach is proposed in Schranzhofer et al. [30], merging pre-computed template mappings (defined at design time) and online decisions that define newly arriving tasks to the processors at run-time. Compared to the static-mapping approaches, obtained results reveal that it is possible to achieve an average reduction on power dissipation of 40 - 45% in realistic software defined radio applications executed on an MPSoC, while keeping the introduced overhead to store the template mappings as low as 1kB.

Another energy-aware approach is presented in Wilderman et al [39]. This approach employs a heuristic that includes a Neighborhood metric inspired by rules from Cellular Automata, which allows decreasing the communication overhead and, consequently, the energy consumption imposed by dynamic applications. Lu et al. [22] propose a dynamic mapping algorithm, called Rotating Mapping Algorithm (RMA), which aims to reduce the overall traffic congestion (take in account the buffer space) and communication energy consumption of applications (reduction of transmission hops between tasks).

In turn, Mandelli et al. [23] propose a power-aware task mapping heuristic, which is validated using a NoC-based MPSoC described at the RTL level, with a clock-cycle accurate ISS describing processors. The mapping heuristic is performed in a given processor of the system that executes a preemptive operating system. Due to the use of a low level description, accurate performance evaluation of several heuristics (execution time, latency, energy consumption) is supported. However, the scope of the work is limited to small systems configurations due to the simulation time. In [22] and [23] only one task is assigned to each PE. A multi-task dynamic mapping approach was also proposed. Singh et al. [31] extends the work, which evaluates the power dissipation as the product of number of bits to be transferred and distance between source-destination pair.

Research in energy-efficient allocation for HPC and cloud systems is still incipient, with existing works addressing only the time and space fragmentation of resource utilisation at a very large granularity (server level), aiming to minimise energy by rearranging the load and freeing servers that are then turned off [4][27].
3. **DYNAMIC RESOURCE ALLOCATION REQUIREMENTS IN DREAMCLOUD**

3.1 **INTRODUCTION**
In order to be applicable to different types of platforms, from embedded to HPC systems, the dynamic resource allocation heuristics developed in the DreamCloud project should satisfy various requirements that are typical for a particular system, as well as take into account properties of the applications to be executed. Thus a number of heuristics should be proposed, each applicable to different systems and applications. The applicability of every resource allocation algorithm should be well defined and described, so that it would be relatively easy to choose the most suitable algorithm for a given task. In particular, at least one heuristics of the types enumerated in the following subsection should be developed. A brief outline of the preliminary analysed algorithms is also provided.

3.2 **VARIETY OF RELEVANT DYNAMIC RESOURCE ALLOCATION HEURISTICS**
The heuristic algorithms enumerated below are preliminary chosen to adequately reflect the diversity of the applications and platforms covered by the DreamCloud project. Since these heuristics have been selected and analysed by the project partners at the early stage of the project progress, the text provided below should be viewed as an early analysis prone to change during further project development, especially as a result of progress made in the course of WP2 (Dynamic Resource Allocation Techniques) and WP3 (Time and Energy Predictability in High Performance and Embedded Cloud Systems), which is consistent with the TOGAF Architecture Development Methodology\(^1\), used in the whole project.

3.2.1 **Control-theoretic-based heuristics**
Algorithms benefiting from control theory are increasingly popular in computer system development. In this approach, a feedback mechanism is used to monitor the capacity of compute resources and QoS levels. They can guarantee a bounded time response, stability, bounded overshoot even if exact knowledge of system workload and service capacity is not available a priori. Thus, in case of careful fine-tuning parameters, they can be successfully applied even to systems with hard real-time constraints, whereas numerous soft real-time system realisations can be found in literature and are also confirmed by the project partners' previous work ([2], [13]). It has been verified that this approach helps to find a trade-off between a few objectives of a workflow management system, e.g. minimal slacks and maximum CPU utilisation. These algorithms are mainly centralised, but for the sake of scalability, the controllers can be situated at different levels in a hierarchical manner. Also it is possible to use so-called gain scheduling, where various conditions may call for different behaviours of the controlled system and, consequently, selecting different values of the controllers' parameters. Although these parameters should be pinpointed in advance, selecting the most appropriate ones for a particular system condition is performed at runtime, leading to maintaining even the hard real-time constraints.

\(^1\) TOGAF™ - The Open Group Architecture Framework - www.opengroup.org/togaf
3.2.2 Evolutionary Algorithms

Evolutionary and genetic algorithms are omnipresent and universal search-based heuristics which can be applied to the dynamic resource allocation. Since this approach is famous for unbounded and unpredictable time overhead, its usage during run-time in both soft and hard real-time systems is not obvious. In the view of the project partners, these algorithms are to be used at design time to evolve acceptable dynamic mappings. During runtime, an agent will chose a pre-computed dynamic allocation being the most appropriate for the current situation. The algorithm realized by the agent is characterised with bounded and low runtime overhead. Consequently, not imposing large computational complexity during runtime, we can cope with multiple fitness functions, e.g. can express a trade-off between power dissipation and response time, and we can even find a task mapping that meets the system’s hard real-time constraints while minimising energy dissipation. As this approach is mainly intended to hard real-time systems, it assumes exact knowledge of the workload and service capacity a priori. However, the limitation of a typical embedded systems in terms of available amount of memory to store the precomputed scenarios, has to be respected.

Some project partners' research on genetic algorithms application to map hard real-time into MPSoC can be found in [28] and [29].

3.2.3 Market-inspired heuristics

Market-inspired heuristics use the guarantees and available capacity provided by low-level heuristics as bids within an auction-like allocation process. For each task, a value curve is generated, which assigns appropriate benefits to task completion in particular time. This approach seems especially tempting in case of changing workload dynamics. It enables users to submit low-value (e.g. best-effort services) jobs speculatively, maintaining high priority of real-time jobs. Since the dynamic resource allocation overhead depends polynomially on the number of tasks, this heuristic can be applied directly in soft real-time systems, whereas in hard real-time it has to be used at design time to generate acceptable dynamic mapping sets. Similarly to evolutionary approach, an agent of relatively low computational complexity would then select the most appropriate assignments. The agents can be organised in a hierarchy to improve scalability of the approach.

3.2.4 Swarm-Intelligence inspired heuristics

The bio-inspired heuristics imitate particular biological system. Among bio-inspired algorithms, a class drawing inspiration from swarm intelligence can be singled out. Multiple agents in these algorithms follow a number of relatively simple rules, which results in their collective behaviour. These heuristics are worth analysing in the project since they are (usually) distributed and self-organising, whereas algorithms implemented in each agent are of low computation complexities and can be easily parallelised. To date, project partners analysed one particular algorithm of this kind, namely Pheromone-Signalling-Based Load Balancing Algorithm. Although it is difficult to predict the final system’s parameters, and in particular to guarantee of meeting any constraints, this algorithm behaves promising in numerous situations according to the already conducted experiments (some results are presented in [7]). Despite being characterised with low computation and communication overheads, it
copes easily with changing workload dynamics. Since each node uses only information available locally, this algorithm scales well and avoids generating any hot-spots.

3.3 REQUIREMENTS LIST

Regarding dynamic resource allocation algorithms, we list the following requirements for the proposed set of heuristics, so that they are universal enough to be applicable to the considered types of applications and platforms.

3.3.1 Objectives of dynamic resource management should be configurable

It is necessary for allocating resources to take multiple criteria into account. The most important objectives are: minimal execution time, lowest costs, highest energy efficiency, and maximal thermal imbalance provided that functional correctness and meeting deadlines in hard real-time systems are guaranteed. The importance of particular objectives depends on a particular application and its domain. In HPC applications these objectives are of utmost importance, as parallel efficiency usually depends on the amount of resources available. So while executing a given task on more resources can improve the time to solution, at the same time the total cost and power consumption might increase in a non-linear fashion.

3.3.2 Specified hard real-time constraints shall not be violated

Some tasks, particularly in embedded clouds, must satisfy timing constraints. Failure in meeting these constraints can lead to catastrophic results in terms of life or property losses. One of the forms of guaranteeing meeting these deadlines is to use static scheduling and allocation with known WCET. Since this option can lead to inefficient resource utilization, it seems beneficial to prepare a set of pre-run-time allocations, satisfying the constraints, and choose between them in real-time to fit the actual scheduling to the external situation. None of these scheduling and allocations poses any risk of violating hard real-time constraints. A run-time overhead is negligible, and unused resources may be temporarily allocated to non-real-time tasks.

3.3.3 Dynamic resource allocation shall be used to provide different levels of performance guarantees

Typical real-time workloads are highly heterogeneous in terms of timing requirements and may consist of hard real-time, soft real-time, and best-effort tasks. While meeting of hard real-time deadlines (if present) must be guaranteed by any developed heuristics, deadline misses for soft real-time tasks and response times for best-effort tasks should be minimized to the possible extent. However, a number of developed heuristics within the project will not be intended for workload with hard real-time tasks and thus focus on system utility maximisation, but providing only probabilistic guarantees on meeting time constraints.

3.3.4 The average latency of jobs shall be minimised

An application job may be comprised of hard real-time, soft real-time, and best-effort tasks. Despite the soft real-time and best-effort tasks are characterised with lower priority and the latter include no deadlines, their execution latency should be also minimised by the dynamic resource allocation mechanism. In some platforms, such as
Networks on Chips, the communicating tasks of an application may be placed close to each other to minimize the communication overhead and, consequently, improve the performance of the overall system. The dynamic heuristics should in this case attempt to map the communicating tasks in a close proximity.

3.3.5 The total energy dissipation of jobs shall be minimised

The mapping heuristics shall provide mechanisms to analyse power consumption of application jobs of different levels of performance guarantees. This information should be used to adjust allocation and scheduling policy. An energy-aware heuristics for dynamic task mapping should analyse both the distance and amount of transmitted data between communicating tasks. Power budget for resource management may be configurable.

3.3.6 Communication overhead parameters shall be predictable

Network end-to-end latency shall be predictable. This requirement is particular important in case of hard real-time tasks, as the communication overhead must be taken into consideration for its performance analysis and determining meeting the imposed deadlines. This requirement is also crucial in high performance and embedded cloud system platforms, where application performance depends upon which part of the cloud or platform they are allocated to and which inter-node paths are used for communication between tasks.

3.3.7 Dynamic resource allocation overhead shall be predictable and bounded

Dynamic resource management actions and all its activities shall be predictable in terms of timing, load, power dissipation, etc. This requirement is of primary importance for hard real-time systems and is indispensable for its performance analysis and determining meeting the imposed deadlines. In hard real-time systems, the possibility to disable dynamic resource management shall be provided. The runtime overhead then shall be negligible. Also the dynamic mechanisms shall provide means to react on specific events with minimal overhead, what is necessary for timing critical interrupts executed with maximum performance provided by the hardware. Power consumption for resource management should be predictable. In the case of HPC, the time overhead of the DreamCloud allocation heuristics should be of the same order of magnitude as the local resource manager.

3.3.8 The dynamic resource allocation mechanisms shall cope with dynamic workload

Both the global and the local schedulers (if present) should support multiple workflows simultaneously. This can be achieved by e.g. offering several parallel end-points for each of the submitted workflows. Costs, execution time and other non-functional properties of the workflow should be predictable and configurable by the user.

3.3.9 The dynamic resource allocation mechanisms shall not limit hardware scaling

Mechanisms to configure and re-configure available resource pools may be supported by the scheduler, e.g., for adding a new cluster to the cloud environment. The scheduler should be able to dynamically handle any updates in the infrastructure pool and adjust the dynamic allocation algorithms dynamically upon adding a new (or losing an already
existing) infrastructure resource. In embedded clouds, one technique to move to distributed mapping is to divide the cloud in clusters, and to provide a mapper to each cluster. Agent-based solutions, with two levels of agents: responsible for global and cluster mapping, may be also considered.

3.3.10 **The dynamic resource allocation mechanisms shall cope with limited information about the state of the overall system**

In a large system, gathering information about the state of each of its components can be a complicated and time-consuming task. It is the reason why algorithms able to make an allocation decision based only on information from neighbouring components are of particular importance. On the other hand, such local information is likely to hinder obtaining the performance close to minimal.

3.3.11 **The dynamic resource allocation mechanisms shall respect mapping constraints that restrict the allowed computational unit**

Certain jobs require specific resources to be executed on. An untyped job can be executed over any type of resource, whereas a single-typed job must be executed over a specific type of resource and thus it has to be statically mapped (for example in end user use case definitions safety critical tasks must be executed on specific safety cores). There exist also multi-typed jobs that can be executed over multiple types of resource. The developed heuristic shall guarantee appropriate constraints of the mapping for single-typed and multi-typed job. This feature is of very high importance in the HPC environment, where jobs often can only utilize specific type of resources within a cloud or cluster. This includes certain programs only running on CPU nodes, specific types of accelerators (e.g. only NVIDIA or Xeon Phi, but not both) and even requirements to the runtime environments available on the systems.

3.3.12 **The dynamic resource allocation mechanisms shall consider cost, runtime and power efficiency for different type of resources available to a multi-typed job**

In the HPC environment execution time, cost of execution and power efficiency highly depend on the type of resource being used. I.e. a job utilizing a different type of resource usually has effects on the time to solution, parallel efficiency, overall costs and power efficiency. For example a job being able to run on GPGPU resources might be able to generate a result faster, than on a CPU resource, but at a higher cost and worsened power efficiency. The allocation mechanism shall be able to take these varying costs for multi-typed jobs into account and handle them according to the objectives described in 3.3.1.

In Table 1, the requirements described in subsection 3.3 are enlisted together with the dynamic resource allocation heuristic initially analysed by the project partners (briefly described in subsection 3.2). In the last column the Work Packages, which address the particular requirements, are enlisted.
Table 1: Dynamic resource allocation heuristics, their requirement fulfilment (CT denotes Control-theoretic-based, EA - Evolutionary, MI - Market-inspired, and SI - Swarm-Intelligence-inspired heuristics), and the related Work Packages.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>CT</th>
<th>EA</th>
<th>MB</th>
<th>SI</th>
<th>Work Package (Task)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1 Objectives of dynamic resource management should be configurable</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>WP2 (T2.3), WP3 (T3.3, T3.4)</td>
</tr>
<tr>
<td>3.3.2 Specified hard real-time constraints shall not be violated</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>WP2 (T2.1)</td>
</tr>
<tr>
<td>3.3.3 Dynamic resource allocation shall be used to provide different levels of performance guarantees</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td>WP2 (T2.1, T2.2), WP3 (T3.3)</td>
</tr>
<tr>
<td>3.3.4 The average latency of jobs shall be minimised</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>WP2 (T2.1, T2.2)</td>
</tr>
<tr>
<td>3.3.5 The total energy dissipation of jobs shall be minimised</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td>WP2 (T2.3), WP3 (T3.3, T3.4)</td>
</tr>
<tr>
<td>3.3.6 Communication overhead parameters shall be predictable</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>WP2 (T2.1, T2.2), WP3 (T3.3, T3.4)</td>
</tr>
<tr>
<td>3.3.7 Dynamic resource allocation overhead shall be predictable and bounded</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>WP2 (T2.1, T2.2, T2.3), WP3 (T3.3, T3.4)</td>
</tr>
<tr>
<td>3.3.8 The dynamic resource allocation mechanisms shall cope with dynamic workload</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>WP2 (T2.1, T2.2, T2.3), WP3 (T3.3, T3.4)</td>
</tr>
<tr>
<td>3.3.9 The dynamic resource allocation mechanisms shall not limit hardware scaling</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>WP2 (T2.1, T2.2, T2.3), WP3 (T3.1, T3.3, T3.4)</td>
</tr>
<tr>
<td>3.3.10 The dynamic resource allocation mechanisms shall cope with limited information about the state of the overall system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>WP2 (T2.2, T2.3), WP3 (T3.3, T3.4)</td>
</tr>
<tr>
<td>3.3.11 The dynamic resource allocation mechanisms shall respect mapping constraints that restrict the allowed computational unit</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>WP2 (T2.1, T2.2, T2.3), WP3 (T3.1, T3.3, T3.4)</td>
</tr>
<tr>
<td>3.3.12 The dynamic resource allocation mechanisms shall consider cost, runtime and power efficiency for different type of resources available to a multi-typed job</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td>WP2 (T2.1, T2.2, T2.3), WP3 (T3.1, T3.3, T3.4)</td>
</tr>
</tbody>
</table>

All these requirements will be addressed in WP2, where a number of heuristics that can be used to provide different levels of performance guarantees, and that cope with different levels of dynamism on the application workload will be proposed and applied to different high-density systems: embedded many-cores, embedded clouds and HPC clouds. Especially requirements connected with hard-real-time application are the
subject of Task 2.1, whereas Soft Real-Time and Best Effort Dynamic Resource Allocation is covered by Task 2.2. In Task 2.3 heuristics aiming at energy dissipation minimisation are to be developed, together with the techniques whose optimisation objectives of dynamic resource management can be configurable.

In WP3, allocation heuristics suitable for both the high performance and embedded cloud computing domains will be developed. In contrast with the lightweight algorithms from WP2, these heuristics will consider the specific complexities of these domains, including additional issues of communication and dynamism imposed by cloud platforms, and their hierarchy of resources.
4. OPERATING SYSTEM AND VIRTUAL MACHINE SUPPORT

4.1 INTRODUCTION

The dynamic resource allocation heuristics, enumerated in subsection 3.2, impose some requirements on the underlying virtual machine and operating system. At this abstraction level, the internal algorithm is of no importance provided that it generates appropriate output from input. It can be then treated as a black box that transforms an input to an output. The virtual machine or operating system should guarantee that the appropriate input data is available and that the generated output can be used to perform the proper resource allocation.

Usually to perform a resource allocation decision we can rely on various metrics. The adequate selection of input and output of the resource allocation heuristic should take into account their availability, as some metrics, for example task arriving time, may be not provided by the operating system. The scheduling algorithms should rely on the performance metrics provided by the monitoring infrastructure tools and services. For a majority of the dynamic allocation heuristics, mechanisms should be provided to monitor time latency between input and output timestamps and further to determine meeting deadlines, execution time, and communication latencies according to the heuristics' needs.

An operating system should also guarantee an appropriate level of responsiveness to the decisions made by the heuristics, as well as update the values of the metrics used as inputs in the algorithm frequent enough for the particular application.

The platform should support scheduling on distributed-memory infrastructure resources. In this case network latency between different processing units and other resources shall be taken into account. It is important to provide to the heuristic algorithm realistic data about system workload, service capacity, worst-case execution time and average end-to-end response times. The context switching time should be bounded and predictable, especially for hard real-time systems. If the target platform supports task migration, it is necessary to take into account the overhead for the migration, e.g. connected with cache warm-up, potentially increased or decreased communication latency, and necessary routing updates. Availability, accuracy and frequent updates of these metrics influence the quality of the final dynamic resource allocation.

Finally, some mechanisms for altering task priority and killing task execution should be provided. Also, the crucial features from the RT-POSIX 1003.1b standard, such as preemption using task priorities, control of priority inversion, high resolution timers, scheduled interrupt handling, and inter-task communication with bounded and predictable delay, shall be available.

4.2 RELATION WITH TASK SCHEDULING

A task mapping process is comprised of the resource allocation and task scheduling. The former is covered in the DreamCloud project, the latter is treated as the part of the underlying operating system. To follow the resource allocation requirements, the influence of task scheduling has to be considered to check if the assumed goals are met.
All of the analysed algorithms assume the presence of a common task queue, which is used by a global scheduler. The Swarm-Intelligence-inspired heuristic mentioned prior in this document is the only proposed technique that uses a peer architecture, the remaining algorithms assume the master/slave approach. In a peer architecture, each processing unit performs self-scheduling from the common task queue. It is the role of the operating system to ensure that a task is executed by only one processing unit at a time and that no process has been denied by all the processing units.

In the remaining dynamic task allocation approaches, a master/slave scheduling architecture is assumed. The resource allocation process is executed on a particular processing unit. Its role is to send the tasks to be executed to other (slave) processing units, putting them into the task queue of a particular processor. Since scalability may be an issue in this approach, it should be possible to introduce an intermediate level in the queuing system and, consequently, form hierarchy dependencies between allocating master units.

The process dispatching, i.e., selecting the actual process to run, is also a part of the scheduling algorithm and, as such, is outside the scope of the DreamCloud project. It is assumed that task scheduling is performed in a preemptive priority-based manner with small and bounded overhead.

4.3 INTERFACE OF RELEVANT DYNAMIC RESOURCE ALLOCATION HEURISTICS

The heuristic algorithm types enumerated in subsection 3.2 are repeated in Table 2. For each of the types, the inputs and outputs of their realisations found in the literature and (preliminarily) analysed by the project partners have been listed. As is stated in subsection 4.2, the metrics, enlisted in the Run-time Input column, should be delivered by the underlying operating system or virtual machine in order to proceed with the dynamic resource allocation stage. The majority of algorithms applicable in hard real-time systems require providing information about application to be executed (mainly its tasks’ worst case execution time), during design time, which is particularly important for algorithms with unbounded computation time, such as evolutionary approaches. In case of soft-real-time systems, inaccuracy of WCET data may worsen the resource utilization. However, due to utilisation of real-time metrics (enlisted in the third column in the table) even in that case the assumed output parameters’ quality of dynamic task allocation should be achieved.

Since the DreamCloud project spans three diverse platform types with substantially different operation systems in specific configurations, some metrics may not be available or not accurate enough to be used in a dynamic resource allocation process. In these situations different metrics and/or different heuristics developed during the course of the DreamCloud project should be applied.
### Table 2: Inputs and outputs of various dynamic resource allocation algorithms.

<table>
<thead>
<tr>
<th>Type</th>
<th>Design-time Input</th>
<th>Run-time Input</th>
<th>Output</th>
<th>Bibliography Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control-theoretic-based heuristics</td>
<td>WCET for tasks with real-time requirements (mandatory for hard real-time, optional for soft-real time systems)</td>
<td>CPU Utilization</td>
<td>Task Rate (for systems with flexible task rates which can be adjusted without causing application failure)</td>
<td>[20], [37]</td>
</tr>
<tr>
<td></td>
<td>WCET for tasks with real-time requirements (mandatory for hard real-time, optional for soft-real time systems)</td>
<td>Scheduler queue fill level, CPU utilization</td>
<td>Admission control, service can be granted (request admitted) or denied (request rejected)</td>
<td>[1]</td>
</tr>
<tr>
<td></td>
<td>WCET for tasks with real-time requirements (mandatory for hard real-time, optional for soft-real time systems)</td>
<td>Connection delay ratio of task type classes</td>
<td>Process budget (i.e., the number of server processes allocated to a certain class in the sampling period; increasing the process budget of the class leads to a shorter connection delay for this class)</td>
<td>[1]</td>
</tr>
<tr>
<td></td>
<td>WCET for tasks with real-time requirements (mandatory for hard real-time, optional for soft-real time systems)</td>
<td>Scheduler queue fill level</td>
<td>CPU cycles allocated to the consumer</td>
<td>[34]</td>
</tr>
<tr>
<td></td>
<td>WCET for tasks with real-time requirements (mandatory for hard real-time, optional for soft-real time systems)</td>
<td>Slack of tasks</td>
<td>Admission Control of a few priority level tasks (with different computational costs)</td>
<td>[13]</td>
</tr>
<tr>
<td></td>
<td>WCET for tasks with real-time requirements (mandatory for hard real-time, optional for soft-real time systems)</td>
<td>Deadline miss ratio</td>
<td>Admission control, Service Level control</td>
<td>[19]</td>
</tr>
<tr>
<td></td>
<td>WCET for tasks with real-time requirements (mandatory for hard real-time, optional for soft-real time systems)</td>
<td>CPU Utilization, Task rates</td>
<td></td>
<td>[37]</td>
</tr>
<tr>
<td>Method</td>
<td>Requirements</td>
<td>Deadline Miss Monitor</td>
<td>Source</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------</td>
<td>-----------------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>Evolutionary Algorithms</td>
<td>Worst case tasks computation time</td>
<td>Fully schedulable task mappings (computed during design time)</td>
<td>[15]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WCET for tasks with real-time requirements, their periods, priorities, communication schema</td>
<td>Fully schedulable task mappings (computed during design time)</td>
<td>[29]</td>
<td></td>
</tr>
<tr>
<td>Market-inspired heuristics</td>
<td>WCET for tasks with real-time requirements, task benefit densities (release time, beginning of optimal execution, soft deadline, hard deadline)</td>
<td>Fully schedulable task mappings</td>
<td>[17]</td>
<td></td>
</tr>
<tr>
<td>Swarm-Intelligence inspired heuristics</td>
<td>Local weighted average of CPU utilization</td>
<td>Binary decision of task admission</td>
<td>[7]</td>
<td></td>
</tr>
</tbody>
</table>
REFERENCES


