<table>
<thead>
<tr>
<th>DELIVERABLE NUMBER</th>
<th>D5.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELIVERABLE TITLE</td>
<td>Resource Utilization and Allocation – Intermediate Version</td>
</tr>
<tr>
<td>RESPONSIBLE AUTHOR</td>
<td>ISMB</td>
</tr>
<tr>
<td>GRANT AGREEMENT N.</td>
<td>688386</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------</td>
</tr>
<tr>
<td>PROJECT REF. NO</td>
<td>H2020- 688386</td>
</tr>
<tr>
<td>PROJECT ACRONYM</td>
<td>OPERA</td>
</tr>
<tr>
<td>PROJECT FULL NAME</td>
<td>LOw Power Heterogeneous Architecture for Next Generation of SmaRt Infrastructure and Platform in Industrial and Societal Applications</td>
</tr>
<tr>
<td>STARTING DATE (DUR.)</td>
<td>01/12/2015</td>
</tr>
<tr>
<td>ENDING DATE</td>
<td>30/11/2018</td>
</tr>
<tr>
<td>PROJECT WEBSITE</td>
<td><a href="http://www.operaproject.eu">www.operaproject.eu</a></td>
</tr>
<tr>
<td>WORKPACKAGE N.</td>
<td>TITLE</td>
</tr>
<tr>
<td>WORKPACKAGE LEADER</td>
<td>IBM</td>
</tr>
<tr>
<td>DELIVERABLE N.</td>
<td>TITLE</td>
</tr>
<tr>
<td>RESPONSIBLE AUTHOR</td>
<td>Dr. Alberto Scionti</td>
</tr>
<tr>
<td>DATE OF DELIVERY (CONTRACTUAL)</td>
<td>31/05/2017 (M18)</td>
</tr>
<tr>
<td>DATE OF DELIVERY (SUBMITTED)</td>
<td>31/05/2017</td>
</tr>
<tr>
<td>VERSION</td>
<td>STATUS</td>
</tr>
<tr>
<td>NATURE</td>
<td>R(Report)</td>
</tr>
<tr>
<td>DISSEMINATION LEVEL</td>
<td>PU(Public)</td>
</tr>
<tr>
<td>AUTHORS (PARTNER)</td>
<td>A. Scionti (ISMB), P. Ruiu (ISMB), J. Nider (IBM), M. Rapoport (IBM), S. Barbier (NEAVIA), G. Urlini (ST)</td>
</tr>
<tr>
<td>VERSION</td>
<td>MODIFICATION(S)</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
</tr>
<tr>
<td>0.1</td>
<td>First Draft</td>
</tr>
<tr>
<td>0.2</td>
<td>Revised structure and filled contents</td>
</tr>
<tr>
<td>0.3</td>
<td>Reviewed by consortium’s partners (NEAVIA, ST)</td>
</tr>
<tr>
<td>1.0</td>
<td>Final version</td>
</tr>
<tr>
<td>PARTICIPANTS</td>
<td>CONTACT</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>STMICROELECTRONICS SRL</td>
<td>Giulio Uralini</td>
</tr>
<tr>
<td>IBM ISRAEL SCIENCE AND TECHNOLOGY LTD</td>
<td>Email: <a href="mailto:Giulio.urlini@st.com">Giulio.urlini@st.com</a></td>
</tr>
<tr>
<td>HEWLETT PACKARD CENTRE DE COMPETENCES (FRANCE)</td>
<td>Joel Nider</td>
</tr>
<tr>
<td>Email: <a href="mailto:joeln@il.ibm.com">joeln@il.ibm.com</a></td>
<td></td>
</tr>
<tr>
<td>NALLATECH LTD</td>
<td>Gallig Renaud</td>
</tr>
<tr>
<td>Email: <a href="mailto:c.petrie@nallatech.com">c.petrie@nallatech.com</a></td>
<td></td>
</tr>
<tr>
<td>ISTITUTO SUPERIORE MARIO BOELLA</td>
<td>Olivier Terzo</td>
</tr>
<tr>
<td>Email: <a href="mailto:terzo@ismb.it">terzo@ismb.it</a></td>
<td></td>
</tr>
<tr>
<td>TECHNION ISRAEL INSTITUTE OF TECHNOLOGY</td>
<td>Dan Tsafrir</td>
</tr>
<tr>
<td>Email: <a href="mailto:dan@cs.technion.ac.il">dan@cs.technion.ac.il</a></td>
<td></td>
</tr>
<tr>
<td>CSI PIEMONTE</td>
<td>Vittorio Vallero</td>
</tr>
<tr>
<td>Email: <a href="mailto:Vittorio.vallero@csi.it">Vittorio.vallero@csi.it</a></td>
<td></td>
</tr>
<tr>
<td>NEAVIA TECHNOLOGIES</td>
<td>Stéphane Gervais</td>
</tr>
<tr>
<td>Email: <a href="mailto:s.gervais@lacroix.fr">s.gervais@lacroix.fr</a></td>
<td></td>
</tr>
<tr>
<td>CERTIOS BV</td>
<td>Frank Verhagen</td>
</tr>
<tr>
<td>Email: <a href="mailto:frank.verhagen@certios.nl">frank.verhagen@certios.nl</a></td>
<td></td>
</tr>
<tr>
<td>TESEO SPA</td>
<td>Stefano Serra</td>
</tr>
<tr>
<td>Email: <a href="mailto:s.serra@teseo.clemessy.com">s.serra@teseo.clemessy.com</a></td>
<td></td>
</tr>
<tr>
<td>DEPARTEMENT DE L'ISERE</td>
<td>Olivier Latouille</td>
</tr>
<tr>
<td>Email: <a href="mailto:olivier.latouille@isere.fr">olivier.latouille@isere.fr</a></td>
<td></td>
</tr>
</tbody>
</table>
### ACRONYMS LIST

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECRAE</td>
<td>Efficient Cloud Allocation Engine</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>ISA</td>
<td>Instruction Set Architecture</td>
</tr>
<tr>
<td>KB</td>
<td>Knowledge Base</td>
</tr>
<tr>
<td>TOSCA</td>
<td>Topology and Orchestration Specification for Cloud Applications</td>
</tr>
<tr>
<td>HOT</td>
<td>HEAT Orchestration Template</td>
</tr>
<tr>
<td>DBMS</td>
<td>DataBase Management System</td>
</tr>
<tr>
<td>LUT</td>
<td>LookUp Table</td>
</tr>
<tr>
<td>ILO</td>
<td>Integrated Lights-Out</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>ES</td>
<td>Evolution Strategies</td>
</tr>
<tr>
<td>ACO</td>
<td>Ant Colony Optimization</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>LC</td>
<td>Linux container</td>
</tr>
</tbody>
</table>

Table 1 List of Acronyms
LIST OF FIGURES

Figure 1 - Conceptual representation of such modern “heterogeneous” data center. ........................................... 12
Figure 2 - Deployment toolchain with application mapping on data center infrastructure. ............................. 13
Figure 3 - Mapping of extended-TOSCA descriptor with infrastructure resources ........................................... 15
Figure 4 - Power efficiency comparison: ECRAE vs basic allocation strategy .................................................. 18
Figure 5 - Resources allocation algorithm – pseudo-code ............................................................................... 19
Figure 6 - Dynamic resource allocation algorithm (PSO) – pseudo-code .................................................. 24
Figure 7 - Page fault handling with cooperative and non-cooperative userfaultfd ......................................... 26
Figure 8 - Example of a set of tasks (cloud applications’ components): each task provides the $\alpha$ value, the CPU and memory load, as well as the associated affinity tag .............................................. 28
Figure 9 - Example of the set of data center nodes available, each with its initial load .................................... 29
Figure 10 - Example of mapping input tasks on the available resources .................................................. 29
Figure 11 -- Results of the task allocation performed by the ECRAE ............................................................. 30
Figure 12 - Starting memcached on src and initiating migration ................................................................. 30
Figure 13 - Completing post-copy migration on dst ...................................................................................... 31
Figure 14 - Verify memcached on dst .......................................................................................................... 31

LIST OF TABLES

Table 1 List of Acronyms ............................................................................................................................... 4
Table 2 List of Actions and Roles ................................................................................................................ 8
ACKNOWLEDGEMENT

This report forms part of the deliverables from a project called “OPERA” which has received funding form the European Union’s H2020 Framework Programme under agreement 688386.

OPERA aims at supporting ambitious challenges on Designing next generation Low Power and Ultra Low Power (ULP) systems, improving energy efficiency in computing by means of heterogeneous architecture, providing a smart and energy efficient solution for the interaction between embedded ULP smart systems and remote small Form Factor Data Centers.
**1 EXECUTIVE SUMMARY**

Work Package 5 (WP5) aims at providing a mechanism to efficiently deploy applications on the data centre resources. The “efficiency” expected from the proposed mechanism strictly concerns the energy consumed by the hardware resources and the relative performance of the various tasks composing applications running on the data center machines. Providing such a mechanism requires to take into account of several elements that influence the runtime execution of such tasks. For instance and to mention few, a model to evaluate the best mapping of tasks with specific host architectures, a mechanism to select the best host among various with the same architecture where to run the tasks, and a way of optimizing the whole data center workload should be integrated.

The aim of task T5.4 is to design a software component capable of deploying application tasks on the most energy-efficient hosts using a energy-aware policy, as well as to provide a way to interact with other components of a Cloud orchestrator. In addition, interactions with other tasks carried out in this WP is necessary to ensure the effectiveness of the whole resource allocation mechanism.

**1.1 POSITION OF THE DELIVERABLES IN THE WHOLE PROJECT CONTEXT**

The activities carried out within task T5.4, whose initial results are summarized in this document, are framed in the WP5 – Optimized Workload Management on Heterogeneous Architecture. Specifically, this deliverable provides the results of the research activity and investigation done by OPERA partners, aiming at developing a software system (ECRAE) to be integrated in the OpenStack orchestration toolchain, which is responsible for the mapping of cloud application components on heterogeneous resources.

The deliverable is in connection with the work done in the WP5, specifically with activities reported in D5.3 and D5.1. Regarding the connection with D5.3, this deliverable specifies the algorithms employed by such software system while D5.3 provides an insight of the interfaces with OpenStack modules that can be used to actually instantiate cloud applications. D5.1 provides the analysis of the TOSCA application descriptor format used by ECRAE to allocate each application components on the most suitable hardware, as well as it provides an analysis of how the workload can be characterized. Other deliverables, such as D5.2 and D.5.5 provide useful information to better tune the efficiency model used by ECRAE to take its decisions, as well as to profiling application and thus how to better characterize them.

Finally, the activities belonging to the task T5.4 provides important and useful results that are at the basis of the activities carried out in WP7, specifically related to the VDI use case. Similarly, activities carried out in WP4 are of interest for this task, since the way energy efficiency is evaluated provides the correct input to drive the decision of the developed ECRAE module.

**1.2 DESCRIPTION OF THE DELIVERABLE**

Specifically, this document reports the results of the initial design phase of such component. Power consumption and host loads are used to build a simple but effective set of strategies aiming at allocating application tasks on the most efficient hosts. Since initial decision are based on a greedy-policy, an optimization process of the infrastructure workload will be integrated. Such part of the designed software component will exploit migration features provided by both traditional virtual machines and modern Linux containers. Applications are described through a standardized application descriptor called TOSCA, which is designed with interoperability in mind. By adapting/extending the TOSCA descriptor, we enable our designed module to allocate resources for the various components of the application.

This document provides a brief introduction of the Cloud orchestration problem, and introduces the open source modular system used in OPERA (i.e., OpenStack) to manage the data center infrastructure. The main OpenStack components are considered, while the integration of the designed module is analyzed in detail. Specifically, the algorithm and the knowledge based used to select at runtime the target host for the task allocation are presented, as well as the optimization procedure to balance the workload. Such optimization procedure is intended to (eventually) migrate virtual machines, as well as Linux containers (LCs). In this perspective, the work done for allowing efficient migration of LCs using CRIU is essential to achieve high level of energy-efficiency in the whole data center infrastructure. The mechanism described allows to put less pressure on the management system, since less data need to be synchronized during the migration process.
1.3 LIST OF ACTIONS AND ROLES

Activities related to D5.4 involved mainly TECH, IBM and ISMB partners. However, also other partners provided useful inputs for driving the activities carried out in T5.4.

<table>
<thead>
<tr>
<th>ACTIVITIES LIST AND PARTNERS ROLES</th>
<th>CERTIOS</th>
<th>CSI</th>
<th>HPE</th>
<th>IBM</th>
<th>ISMB</th>
<th>LD38</th>
<th>NALLATECH</th>
<th>NEAVIA</th>
<th>ST</th>
<th>TECHNION</th>
<th>TESEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOSCA extension and usage</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td>P</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design of the ECRAE module (interface with other modules, integration, algorithms, etc.)</td>
<td>I</td>
<td>I</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of the ECRAE</td>
<td></td>
<td></td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration of ECRAE in the OpenStack environment</td>
<td>P</td>
<td>I</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRIU-based migration mechanism design</td>
<td>I</td>
<td>P</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution to the deliverable organization, writing, and reviewing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 List of Actions and Roles

a)  P = Participating (includes I & R)
b)  I = Input delivery (Includes R)
c)  R = review

ISMB: it is the main contribution for the activities carried out in the task T5.4. ISMB started with the analysis and design of the software module used to schedule cloud applications on the data center infrastructure. ISMB also provided the initial implementation of this software module (ECRAE), as well as it studied the integration of a dynamic re-scheduling policy based on an evolutionary algorithm to better optimize the energy efficiency of the data center. Using inputs provided by CERT and TECH partners, a simple but rather effective (power) efficiency model has been integrated in the ECRAE.

CSI: mainly contributed by providing inputs regarding both the integration of the new module developed in the context of T5.4, as well as it provided inputs on the exploitation and extension of the TOSCA format to trigger the (static) scheduling phase. CSI provided access to the infrastructure, where OpenStack has been installed and it is used to the purpose of integration (see also WP7 – VDI use case).

IBM: contributed to this task, by providing the mechanism to migrate Linux containers, also between nodes with different hardware (ISA, configuration, etc.). IBM also actively contributed to the organization and writing of this document.

TECH: provided inputs for the creation of the (power) efficiency model used by ECRAE to select the most suitable server where to execute specific application components (this information is carried out by the TOSCA file).
CERT: similarly to TECH, CERT provided inputs for the creation of the (power) efficiency model used by ECRAE to select the most suitable server where to execute specific application components (this information is carried out by the TOSCA file).

STM: reviewed the material provided in this deliverable.

NEAVIA: reviewed the material provided in this deliverable.
TABLE OF CONTENTS

1 EXECUTIVE SUMMARY ........................................................................................................................................... 7
  1.1 POSITION OF THE DELIVERABLES IN THE WHOLE PROJECT CONTEXT ................................................. 7
  1.2 DESCRIPTION OF THE DELIVERABLE ............................................................................................................. 7
  1.3 LIST OF ACTIONS AND ROLES ...................................................................................................................... 8

2 INTRODUCTION .................................................................................................................................................. 11
  2.1 CLOUD ORCHESTRATION IN AN HETEROGENEOUS CONTEXT .............................................................. 12
  2.2 OPERA HETEROGENEITY TARGET ............................................................................................................... 14

3 ECRAE: THE EFFICIENT CLOUD RESOURCES ALLOCATION ENGINE .................................................... 15
  3.1 APPLICATION DEPLOYMENT STRATEGY ........................................................................................................ 15
      3.1.1 Knowledge base organization .............................................................................................................. 16
      3.1.2 (Power) Efficiency Model ...................................................................................................................... 16
      3.1.3 Resources allocation engine ................................................................................................................ 18

4 DYNAMIC WORKLOAD SCHEDULING ADAPTATION ..................................................................................... 21
  4.1 STATE OF THE ART .......................................................................................................................................... 21
      4.1.1 Problem statement ................................................................................................................................ 21
      4.1.2 Evolutionary-based algorithms ............................................................................................................. 22
  4.2 INTEGRATION OF THE HEURISTIC .............................................................................................................. 23
  4.3 CONTAINER MIGRATION ............................................................................................................................ 24
      4.3.1 Post-copy vs Pre-copy .......................................................................................................................... 25
      4.3.2 Post-copy Migration Implementation .................................................................................................. 25
  4.4 FINAL REMARKS ........................................................................................................................................... 27

5 EXPERIMENTATION ........................................................................................................................................... 28
  5.1 TASKS DEPLOYMENT EXAMPLE ................................................................................................................... 28
  5.2 MIGRATION EXAMPLE ................................................................................................................................... 30

6 CONCLUSIONS ................................................................................................................................................... 32

7 REFERENCES ......................................................................................................................................................... 33
Work Package 5 (WP5) – Optimized Workload Management on Heterogeneous Architecture – deals with innovative methods for reducing energy consumption and increasing efficiency by scheduling tasks to appropriate compute resources within a data center. Efficient resources allocation requires good understanding of how run-time impacts on the performance and energy consumption of the hardware infrastructure. To this end, WP5 is organized into 5 separate tasks, which aim at: (i) analyzing and characterizing the behavior of cloud applications in order to drive the resources allocation policy towards a more efficient decision–T5.1; (ii) providing a power model to be used by the resources allocation policy–T5.2; (iii) developing a software system able to allocate resources by trying to minimize the overall energy consumption–T5.4 and (iv) capable of interfacing with current orchestration tools–T5.3; (v) measuring energy and performance for different memory-intensive workloads, and then trying to conclude a model for runtime/energy overhead when using different amount of compute resources–T5.5. Specifically, this document refers to the definition, design and implementation of the software system that is in charge of allocating the various software modules composing a cloud application.

Work and research activity carried out in T5.4 is mainly focused on the development of a software system (the “Efficient Cloud Resources Allocation Engine”–ECRAE) that is capable of allocating cloud infrastructure resources to the cloud application modules, depending on two main constraints: first, the application module generally comes with some specific configuration required to correctly run; second, the cloud infrastructure is becoming ever always more heterogeneous. Regarding the first point, this means that ECRAE must select the host nodes in such way to satisfy the minimum configuration, and to minimize at the same time the energy consumption. Regarding the second point, ECRAE should be capable of considering the availability of host with extremely different characteristics, thus a mechanism to evaluate them in terms of performance vs. power consumption must be included. For instance, OPERA will make large use of FPGA cards to accelerate different parts of specific workloads. FPGA are in nature completely different from “processors”, since they do not execute instructions operating on data, but rather they process a stream of data flowing inside the chip. This requires a way to compare the efficiency also of such systems with regards to processors (i.e., CPUs or other accelerators like GPUs). This knowledge should become part of the input of the ECRAE module.

Cloud applications are different in nature with regard to traditional HPC-oriented applications. In fact, Cloud infrastructures are generally optimized to deliver services, which are expected to run for a period of time that is not predictable. For instance, a database used to manage an online storage service (e.g., as in ownCloud application) is expected to be online always (i.e., at least till the service is running or a failure occurs). This makes difficult to base the resources allocation policy operated by ECRAE on the “energy” consumption, while it is more simply to consider the “power” consumed by different host as a good estimator for the energy in the long term. In this document we will discuss the greedy algorithm that uses host power estimation to generate an initial cloud application mapping on the infrastructure resources. Since this mapping could results in a globally unoptimized allocation, a periodic global allocation mapping (i.e., performed considering all the software modules running in the infrastructure) is performed. This step resembles a typical optimization problem, already well studied in literature. To this end, an indication of the possible heuristics will be drawn.

Looking at the optimization process that (must) should be run periodically, it is worth noting that this activity implies to eventually “migrate” software modules from one host to another. Although the ECRAE is agnostic in principle with regards to the type of virtualization technology used to actually deploy software components, a lightweight mechanism is preferable. From this viewpoint, Linux Containers (e.g., Dockers, LXC/LXD, etc.) represent a good solution, since their impact on the host memory is lower than traditional VMs. In this context, a mechanism for facilitating the migration of LXC containers between host exposing a different architecture (i.e., different ISA, memory configuration, etc.) is discussed. On the other hand, the analysis of the impact of workloads on the memory subsystem is presented in deliverables regarding T5.5 (in fact, virtualization systems relying most on specific features provided through the virtual memory). Finally, some example of the allocation and migration provided solutions will be also discussed.
2.1 CLOUD ORCHESTRATION IN AN HETEROGENEOUS CONTEXT

Cloud computing paradigm is based on the availability of a large set of compute, storage and networking resources. These resources, which are in general heterogeneous in nature, need to be controlled in such way user requirements are met and resource usage is maximized. Recently, with the growing demand for more energy-aware systems (reducing energy consumption is essential to sustain the growing demand for computing and storage resources), heterogeneity in data centers has been further increased by introducing different processor architectures and dedicated accelerators. The formers are well represented by the growing presence of ARM-based systems, while the latter are represented by GPUs and recently the support for FPGA devices. In OPERA, we further extended such heterogeneity by introducing in the infrastructure also POWER8 based systems. This is a platform originally intended for high-performance computing machines; however, with the emerging of Cloud services supporting HPC oriented workloads and scientific applications, the availability of dedicated computing nodes becomes of worth. It is worth to highlight, that heterogeneity in data centers extends in several dimensions: resources are heterogeneous because of their inherently different architecture and instruction set (ISA - Instruction Set Architecture), but also at more coarse level when nodes are configured with different amount of resources (e.g., memory, number of cores, storage, etc.).

Moreover, machines configured in the same way and using the same CPU architecture, can be still different each other since CPUs’ families change the features over various generations (e.g., the most recent Intel Xeon processors support AVX-512 instruction set extension which was not available on previous models) as well as the supported clock frequencies. Recently, the heterogeneity landscape has been further extended thanks to the large availability of FPGA acceleration cards. Field Programmable Gate Arrays - FPGAs - are interesting acceleration solution for many cloud intensive tasks, since the chip structure can be re-organized in order to run a specific algorithm as a digital circuit. Thus, instead of executing a flow of instructions, FPGAs process streams of data using a dedicated digital circuit. The interest for including such devices in data centers comes from their high performance/power consumption ratio. Since provided performance are high, the time required to process large bunch of data is reduced, and thus energy consumption is reduced as well. Figure 1 depicts the conceptual representation of such modern “heterogeneous” data center, highlighting the presence of software stack devoted to management.

One of the main task in governing such kind of infrastructure regards the allocation of resources for the different applications, as well as the need of being “energy-efficient”. In order to be “efficient” such allocation should consider different additional factors to get the optimal allocation decisions. To this purpose power consumption of each platform and the relative load are considered good estimators of the relative energy consumption (in this first phase, we consider this two metrics as good estimators for taking correct decision in the resource allocation phase). To our knowledge, current Cloud orchestration tools are designed to be able to abstract as much as possible the data center infrastructure. The drawback of this
process is the lost of energy/power consumption awareness in the allocation process of the infrastructure resources.

OPERA aims at improving current systems and tools used to orchestrate applications in the cloud environment (i.e., specifically, at the infrastructure level). Orchestration generally refers to the automation of the actions required to deploy an application on the cloud infrastructure, in particular it concerns the allocation of resources, the installation of application components on such acquired resources, and the needed configuration of application components and infrastructural elements. In this perspective, the allocation of resources by optimizing the energy efficiency of the cloud infrastructure represents a key element. Although, it is our intention to design and develop a solution that can be used in different contexts (i.e., not locked to a specific vendor existing platform), OPERA selected one popular platform as a reference. Thus, one of the main activities carried out in WP5 is specifically to enhance the OpenStack system in such way it is possible to:

- Phase-1: deploy the application components on the most suitable platform (i.e., by choosing the better processor architecture, amount of memory, storage space, etc.), with the aim of maximizing the energy efficiency;
- Phase-2: periodically rescheduling (i.e., migrating) the application components on the most suitable platform if different load/efficiency conditions arise. For instance, a web-frontend previously running on an ARM-based server can be moved on a X86_64 machine if the load of the ARM machine exceeded a threshold and/or the number of requests to the frontend increased.

In this regard, we refer to the first phase as a static deployment action, while we talk of dynamic (re-)scheduling of the application components in the second phase. In order to perform these two actions, the Cloud system needs to be able to match, for each application component, the most suitable platform based on the indication collected in a Knowledge Base (KB), and to monitor the status of the infrastructure. Actually, the selection of the platform is reduced to the selection of one host server machine exposing desired/required hardware and software features (e.g., a server with a X86 processor, equipped with a certain amount of memory and storage, and running a specific version of the operating system). Moreover, it is necessary that the application can be split into independent components, as well as that each component has been somehow profiled. To manage the whole orchestration process, we rely on a standardized application description format (TOSCA) which encapsulates all the details needed to correctly assign the components on the specific platform, and to correctly configure them.

TOSCA—Topology and Orchestration Specification for Cloud Applications—provides a standardized mechanism to describe how a complex cloud applications is composed (i.e., the set of software modules, such as web interface, specific business logic modules, databases, etc.) and how they should be deployed on the infrastructure (i.e., the host machine configuration to use to run them, along with specific installation scripts). Since it is a standard description mechanism, it is agnostic with respect to specific vendor cloud orchestration tools, so that a translation layer will be used. For instance, in OPERA a TOSCA-to-HOT translation layer (HOT is the internal format used by the HEAT module to describe how the various pieces of software have to be deployed in the infrastructure) is used in order to allow OpenStack agents to correctly instantiate infrastructure resources. Figure 2 briefly depicts the schema of instantiation of cloud applications within our OPERA architecture. Similarly to other vendor-based application descriptor, TOSCA allows the user only to specify the host configuration, but since this should be selected depending on “power/energy” consumption metrics, we provide an extended version of the TOSCA descriptors. Host nodes are described...
by a simple label, which is intended to capture the best affinity of the host machine with the software to be deployed. It will be responsibility of the ECRAE to map such initial affinity with an actual host machine configuration. To this purpose a knowledge base (KB—i.e., a database containing information regarding the infrastructure) is used to track the list of nodes available in the infrastructure, and to know their actual status (i.e., the CPU and memory occupation).

2.2 OPERA HETEROGENEITY TARGET

This section aims at summarizing the variety of hardware devices composing the OPERA architecture (X86, ARM, POWER, FPGA), which represent the target of the ECRAE. Although further computing platforms can be part of a modern cloud infrastructure, we can consider the set used in OPERA as representative of the whole spectrum of systems, ranging from low-power architectures (ARM-based), to conventional ones (X86), to high-performance oriented ones (POWER-based solutions). Figure 1 is also representative of such large heterogeneity adopted by OPERA.

Further, FPGA accelerators are becoming more interesting in the cloud context, thanks to their capability of performing specific tasks with an unheard-of level of efficiency (i.e., the ratio between performance and power consumption). In fact, instead of relying on optimized processor pipelines, as well as on the massive parallelism offered by manycores solutions, they leverage on full-custom circuits. Although, programming FPGAs requires a very large effort, since generally VHDL (or Verilog) code must be written, recently advancement in high-level language compilation toolchains allows broader adoption of such systems. OPERA exploits capability of OpenCL, a standardized accelerator-agnostic extension of the C/C++ language, which is able to drive the compilation and synthesis of the FPGA circuit, without the need of writing complex VHDL code.

Apart conventional and largely used X86 systems, the quest for more performance and less energy consumption lead cloud infrastructures to adopt other processor architectures. To represent such common situation, OPERA also exploits the benefit of embedded ARM-based solutions to deliver high performance in a low power envelop, and POWER-based systems. The former are available by accessing the OPERA-designed FPGA board. Thanks to the Intel Arria-10 SoC, up to 2 ARM-Cortex A9 are available per acceleration board. On the other hand, POWER hosts are capable of more performance compared to traditional X86 nodes. Furthermore, they can exploit dedicated interconnection (CAPI) with FPGA systems to avoid unnecessary data movements since coherence is transparently maintained by the hardware. Such system thus, are more interesting to explore for CPU and memory intensive workloads.

For each of such architectural categories, different configuration can exist inside the data center, and for each of this configurations, several nodes can be available. The goal of ECRAE is to tracking the status of each of these nodes, and mapping new requests for resources (i.e., deploying new cloud application parts) to the most appropriate one.
3 ECRAE: THE EFFICIENT CLOUD RESOURCES ALLOCATION ENGINE

This chapter is devoted to the main resource allocation module (ECRAE) we are developing in OPERA. Its purpose is to apply possibly the best strategy to allocate infrastructural resources in order to minimize the energy consumption. Since energy consumption is directly related to the power consumption of the hosts, the ECRAE strategy will exploit a simple but rather effective power-model to select the host where to deploy cloud applications’ components. The usage of a power model becomes necessary to implement a greedy allocation strategy (see Phase-1, Sec 1.1). Since, greedy allocation does not ensure optimal allocation for the whole set of application components, a second step is necessary. Here, a global optimization algorithms is used to re-schedule all the allocated components with the objective of globally reduce the power consumption. Again, since application components can run indefinitely (e.g., a database is expected to be always online, unless a failure occurs or the service is stopped), the usage of a “energy”-based metric is difficult. As a global scheduling strategy, several heuristics have been studied and proposed on literature. Among various, evolutionary-based algorithms have been demonstrated very good performance: they can converge fast with a (near-)optimal solution (i.e., the allocation of all the applications’ components on the available hosts), and they generally exhibit capability of incorporating historical data, meaning that they continuously learn. A further advantage is their simple structure that is inherently parallel in nature. This makes them also suitable for acceleration on dedicated host multicore.

3.1 APPLICATION DEPLOYMENT STRATEGY

The cloud application is described through the TOSCA descriptor. The ECRAE module is responsible to select the most appropriate node where to deploy the application components and to interact with the other OpenStack modules to instantiate these components (more information on the TOSCA format and the comparison with other application descriptors is provided in D5.1 and D5.3).

Here, we aim at summarizing how the modified TOSCA descriptor is used to trigger the selection of the specific nodes in the data center infrastructure. TOSCA provides a hierarchical description of a generic cloud application: for each software component, the set of scripts to manage the installation and the main functionalities exposed by the component is provided, along with the set of requirements. Requirements generally also deal with the underlying components needed to be correctly installed and run. For instance, a MySQL database requires the installation of a MySQL DBMS on one node, as well as a specific Operating System. Thus, the TOSCA descriptor provides also information regarding the dependencies among the components.
The last elements in the hierarchy is represented by features of the hosts. Here, in OPERA we propose to use a generic “tag” to describe the **affinity** of the software components and the host features. Such affinity is representative of a possible configuration that is evaluated as the most suitable for the execution of the specific component (e.g., a “big-memory” tag could be used to represent the configuration of a big memory machine, which is well suited for in-memory database operations). The correspondence between the affinity expressed in the TOSCA descriptor (i.e., a file following a YAML-based syntax) and the possible nodes’ configurations is provided by the **Knowledge Base** (KB). Here, for each configuration, the list of nodes which provide that configuration is also available, along with the actual CPU load, memory load and the maximum power dissipation. Such information are used to rank the nodes according to the ECRAE policies. Once the node that better fits with the ECRAE policies has been selected, the corresponding full configuration is used to replace the affinity element in the TOSCA descriptor. The result of such selection process for each application component is a fully-compliant TOSCA description file, which can be transformed by the TOSCA-to-HOT module into a HEAT-compliant description. The entire process is shown in figure 3.

It is interesting to note that, although the affinity of the application software component with a specific host configuration is a static information provided in the TOSCA description, it is interesting as a future direction of investigation, finding and integrating a mechanism able to allow the orchestration system to “automatically learning” which is the best affinity mapping. In this perspective an initial allocation is provided, but over the time the system could automatically learn which is the best configuration to use. From this perspective, also the affinity mapping would become a dynamic parameter, that allows the OPERA architecture to better adapt to the working conditions.

### 3.1.1 Knowledge base organization

This section summarizes the organization of the Knowledge Base used to map TOSCA tags with corresponding node configurations.

The KB is essentially a (relational) database storing a set of structured information, organized into a set of tables. At its basis, it should work as a lookup table (LUT), where tag elements are used to extract a list of possible configurations. To this purpose, one table is used to associate tags (i.e., affinity) with the specific corresponding platform configuration. On the other hand, a second table provides the information of each node in the data center that matches the specific configuration. In particular, the average power consumption, and the current system load (CPU and main memory) is exported. These information are collected from an external component (e.g., Ceilometer module in the OpenStack environment, Carbon/Graphite, etc.). Information gathered from this external module are loaded in the database and periodically updated, to reflect the actual status of the data center infrastructure, and can be further extended to include more details on the host platforms. Information can include also the operating system running on the specific host (and its version), as well as a detailed description of the processor architecture in use (i.e., architecture, number of cores, frequency, etc.). Finally, in order to correctly select the execution node, all these information are combined into a simple ranking model.

### 3.1.2 (Power) Efficiency Model

The most critical element in the selection of the actual node for executing a cloud application component is the model used to rank the nodes belonging to the data center infrastructure. On the other hand, the main objective and contribution of the OPERA project is the reduction of the energy consumption in the data centers, and its consequently energy-efficiency improvement. One point to keep into consideration is the relation between energy consumption ($E$) and power consumption ($P$).

The power $P$ refers to the instantaneous energy consumed by a system and generally varies over the time (this means that power consumption is described as a function of the time), thus the energy consumed by a system can be computed as the integral of the power consumption on a given period of time:

$$ E = \int_{t_0}^{t_1} P(t) \, dt $$
The power consumed by a server machine depends on several factors; however, we can assume that it is mainly influenced by the consumption of the main components such as CPU, memory, storage and network activity. Power consumption of the CPU and the memory is mainly due to the pressure on these components, i.e., how much the software running stresses these components. Since the load generally changes over the time, thus also the power consumed by the CPU and memory (as well as other components) changes. Another important aspect to take into consideration is the power consumption in the idle state. In literature [1, 2, 3] has been well documented that a conventional server machine, especially if not properly designed, may consume a large amount of power (and thus of energy) even in the idle state. With such state, we refer to a powered machine that is not performing any task. Specifically, has been observed that the power consumption in the idle state can reach up to 70% of the maximum power consumption. Some techniques have been proposed to avoid idle servers to waste energy, by sleeping components and waking up them once new request are received [4, 5]. Such techniques can be of help too in the OPERA context. Finally, it is also to worth noting that for most of the cloud application components is not easy to foresee a duration of the execution (e.g., it is not possible to define for how much time a database should run, since it is expected to be always accessible unless a failure or the service interruption). Given all these consideration we investigated on the best way to rank host node at run-time depending on their current load status and the power consumed.

Given the above consideration, we elaborated a simple but still effective model for ranking the nodes exposing a given configuration. Since we cannot measure the time for which the application software will run, we assume that this software increase the CPU and memory load for a given quantity. Such quantity (Cᵢ –represents the CPU load increase expressed as a percentage, Mᵢ –represents the memory load expressed as a percentage) is measured as the average increase generated by the execution of that application using the host machine in different working conditions. The following equation allows to emit a score value (R) for the node:

\[ R = (\alpha \cdot C_i + (1 - \alpha) \cdot M_i) \cdot P \]

The score \( R \) is the weighted measure of the current power consumption \( P \) of the node (the power weighted value is biased by the power consumption of the nodes in idle state, so that the \( P \) value is given by the power consumption in idle incremented by the fraction due to the machine load), where the weight is expressed by a linear combination of the current CPU load \( C_i \) and the memory load \( M_i \) (both represents the load increase as a percentage; e.g., setting \( C_i = 0.15 \) means that the application will increase the host CPU usage by 15%). The linear combination is obtained by weighting these two load factors with the \( \alpha \) parameter, which allow to express how much the application is CPU intensive or memory intensive. This parameter is part of the information associated to the affinity tag. For instance a database which is expected to perform several transactions, could be associated to a low CPU load and a high memory load (e.g., setting \( \alpha = 0.25 \), the load on the memory would be equal to 75%). Thus, \( \alpha \) can be used to tune the linear combination: for instance, in some host configuration would be more critical to reduce the CPU consumption rather than the memory one (e.g., high-performance chips like POWER). If any specific tuning is required, \( \alpha \) is set (by default) to 0.5.

Power consumption \( P \) is obtained as a measure of the average power consumption of the host platform in different working conditions. Averaging the power consumption allows to capture the typical power profile of the host system. Since this value is read from the KB, a mechanism to periodically update it can be introduced to better reflect real machine behavior (i.e., the average power can be periodically computed by sampling the host power consumption and calculating the average of the set of captured samples –e.g., a sample every 5 minutes). However, such mechanism requires the availability on the host nodes of a hardware power monitor and an interface to query it (e.g., the ILO interface).

For instance, let’s consider two nodes belonging to the two flavours associated to a given affinity tag. For the sake of simplicity we can assume two X86 nodes, each in the idle state, but with different average power consumption: we assume node_1 consuming up to 100W (i.e., assuming 65% of idle power consumption that is equal to 65W), node_2 consuming up to 130W (i.e., assuming 65% of idle power consumption that is equal to 84.5W). Let’s assume to schedule two tasks loading the nodes by 45% each (i.e., the cpu load and memory load are assumed equals to 0.45, and using \( \alpha = 0.5 \)). Given this premise, the basic allocation policy (i.e., assigning the task to the less loaded node) lead to a higher power consumption, as reported in the following figure (figure 4).
In fact, when the first task is selected, the two nodes are in the idle state and both the strategies allocate the task to node_1 (i.e., here we assume node_2 belonging to the alternative flavour of the KB). At this point, the power consumption of node_1 increases up to 80.75W, with an overall power consumption equals to 165.25W. On the other hand, the second task is allocated differently. ECRAE ranks the node depending on their weighted power consumption, thus selecting node_1 also for the second task (although the node_2 is less loaded). This provides further 15.75W of power consumption (with a 0.9 node’s load). Conversely, basic allocation strategy selects the node with the lowest load, leading the node_2 to be selected. In that case, the execution of second task on node_2 provides 20.475W of power consumption, leading to an overall power consumption of 185.72W.

3.1.3 Resources allocation engine

The resources allocation engine (ECRAE) uses the information provided by the power efficiency model described in the previous section to rank all the nodes that match with the configuration associated to the affinity tag. In figure 5, the main algorithm used by ECRAE to select the node for running a specific application component is provided as a pseudo-code. Here, we explain the algorithm in detail.
The first step is to extract the information related to the application component to allocate (lines 3–4). This information regards the increment in terms of CPU and memory loads (as a percentage) and the $\alpha$ parameter. Also the affinity tag is extracted from the TOSCA description. Given the affinity tag, in lines 5 the corresponding configurations (affinities) are extracted. Here, we assumed that the first configuration (affinity_1 in the algorithm) represents the best match with the requirements of the application component; however, an alternative configuration can be exploited (affinity_2 in the algorithm). Given the effective configurations, the algorithm extracts the list of nodes in the data center that has those configurations (lines 6–7), then it creates an empty list that associates to each node the R score. In lines 9–13 a loop is used to create such list: for each node with affinity_1 configuration the CPU load, memory load and average power consumption is used to calculate the R score. In line 14 the scores are sorted, so that the first element should be the best candidate. This candidate is saved in line 15, in case no better solution is found.
From line 16 to line 24 the actual candidate node is searched. To this purpose, each node is extracted from the list. If the increment in the CPU and memory load does not exceed a given threshold (here we used a threshold set to 95%) the node is assumed as the best candidate and the search is stopped. Otherwise, the node is removed from the list (so the loop is interrupted if the list becomes empty) and a new node is evaluated. In case none of the nodes is able to satisfy the selection conditions, the node is searched in the list of nodes with affinity_2 configuration. From line 25 to line 40, a similar procedure is applied. Nodes are evaluated to extract the R score (passing their actual status as CPU and memory loads, and the average power consumption). The list of nodes is then sorted. A loop (lines 33–40) allows to evaluate the nodes to see if the increment in the CPU and memory load does not exceed a given threshold (here we used a threshold set to 95%). If one of the nodes satisfies this condition, then the search loop is stopped, and the corresponding node is returned for the allocation (lines 41–42). Interestingly, if no one of the nodes satisfies the condition on the threshold, then initial best candidate saved in line 15 is returned (lines 43–44).

It is worth to note, that the strategy set here is a greedy strategy that is not able to ensure that a minimum number of nodes are switched on. To avoid powering on a machine that is not running, a global optimization strategy must be put in place (see section 3). This optimization strategy has the main objective to minimize the number of active servers, as well as to reduce the overall power consumption. In fact, as previously motivated, the power consumption of an idle (or even very low loaded) machine can reach up to 70% of its maximum power consumption. Finally, as future activities planned for this task (T5.4), we will investigate on alternative strategies to allocate resources in the deployment phase (see phase-1, section 1.1).
4 DYNAMIC WORKLOAD SCHEDULING ADAPTATION

This chapter describes the integration of an heuristic to periodically redistribute allocated software modules on various nodes. Here, an introduction to the problem is provided. Also, we motivate the usage of a migration-aware container mechanism (i.e., CRIU) as a supporting technology for dynamic reallocations. Given the large number of different resources available in a data center and the dynamicity of the workload that has to use such resources, the problem of correctly assigning them over the time is well represented by an optimization problem. Generally, due to the nature of the assignment, the problem resembles a “bin-packing problem” (BPP), which has been well studied in literature. Although a deterministic algorithm to solve it (i.e., to provide the optimal solution) can be provided, some further constraints limit its usage in practice. Nevertheless, the computational complexity is too large to consider real usage of such kind of algorithms in practice. In fact, BPP has been demonstrated to belong to the class of NP-Hard problem. To practically solve large instances of the problem in a limited amount of time, and obtaining a near-to-the optimal solution, heuristics can be applied. Heuristics allow to find good candidate solutions (eventually they can provide the optimal solution), without any guarantees of the optimality of such solutions.

As stated in the section 2, the greedy strategy used to initially allocate resources for a given software component of the cloud application, may lead to a sub-optimal allocation, if we consider the energy efficiency of the whole infrastructure. Hence, a periodic redistribution of the load in the host servers is required. Based on a heuristic, such procedure is responsible to compute the optimal resource allocation, and then to drive migration of the VMs and/or containers. To this end, in the following both the discussion and presentation of a possible heuristic is given, as well as a solution for efficiently moving containers.

4.1 STATE OF THE ART

Workload scheduling is an optimization problem that is NP-Hard, and can be formulated (actually, the function to be optimized can slightly change; e.g., it can include the energy cost or other forms of costs). as follows: Given a set of different objects, each with a volume \( S_i \), the objective is to assign as much as possible objects to a container (bin) that as a finite volume \( V \). In the specific context of a data center, the reduction of the energy cost (and thus the reduction of the power consumption cost) becomes important, especially if we consider that large amount of energy is consumed by machines running in (near-)idle state. Given this premise, the objective of the heuristic is the minimization of the number of running machine (bins), and also the whole power consumption of the data center (i.e., the heuristic should try to consolidate as much as possible the workload on the minimum number of active hosts). To this point, the workload is characterized by the load (in terms of CPU, memory, and eventually other resources of the host servers) that each software component (here referred to as task) apply to the selected host. Also, it is important to note that such component should maintain as much as possible the affinity expressed initially to not reduce performance, and in general to guarantee their correct execution. A possible solution to the last constraint can be to use as the set of possible target hosts, only the nodes that has the correct affinity.

4.1.1 Problem statement

Given the set \( S \) of physical servers \( s_i \) belonging to the same affinity profile (each with a specific capacity in terms of CPU and memory availability -- offer \( (V_i) \)), and given the set \( T \) of tasks \( t_j \), each requiring a machine with the specific affinity profile of servers in \( S \) (similarly to the servers, also tasks are characterized by the load in terms of CPU and memory usage they need to run -- request), the objective is to minimize the number of active servers in \( S \): 

\[
\text{OF: } S = \min (\sum_{i=1}^{S} y_i)
\]

subject to:

\[
\begin{align*}
\min (S) &\geq 1, \\
\sum_{j=1}^{T} t_{ij} x_{ij} &\leq y_i V_i, \quad \forall i \in \{1,\ldots,|S|\}, \\
\sum_{i=1}^{S} x_{ij} & = 1; \quad \forall j \in \{1,\ldots,|T|\}, \\
x_{ij}, y_i &\in \{0,1\}, V_i \in [0,1], \quad t_j \in [0,1]
\end{align*}
\]
where \( y_i \) is set to 1 if the server \( s_i \) is used, and \( x_{ij} \) is set to 1 if the task \( j \) is assigned to the server \( i \), and where \( t_j \) is the relative size of the task \( j \) (i.e., the load in terms of CPU and memory it provides to the machine).

The objective function \( \text{OF} \) can be further expanded by also requiring that the overall power consumption of the active servers (\( y_i = 1 \)) is minimized, thus the new function to optimize becomes:

\[
\text{OF}: \min \left( \sum_{i=1}^{|S|} y_i + \sum_{i=1}^{|S|} P_i y_i \right)
\]

where \( P_i \) is the power consumption of the server \( i \), once loaded with the assigned tasks (since the minimization of the overall power consumption depends on the number of active machines, we will investigate also if only the second term of the OF equation is enough to lead to an effective allocation).

### 4.1.2 Evolutionary-based algorithms

Several heuristics have been proposed in literature to fast and efficiently compute a solution for the above stated problem. The simplest greedy strategy is called “first-fit” [5, 6], and it basis the allocation decision on a simple rule: the strategy start consuming the server’s resources till the task size is enough small to fit in, one server at a time. The solution provided with this strategy are generally poor in terms of quality, so some variant has been proposed too (e.g., “best-fit” strategy [7, 8]). However, interestingly, due to its simplicity such strategy is largely adopted by cloud provider to allocate resources.

Among the possible algorithms, the class of “evolutionary-based” has been studied as a way of providing high-quality solutions for large instance of complex optimization problems. Such kind of algorithms provide general advantages over the others, by mimicking the temporal evolution of a system:

- They are generally “population based” approaches, so they are parallel in nature;
- They can conveniently exploit modern multi-/manycore solutions;
- They are based on simple rules governing the interaction among the candidate solutions (temporal evolution).

Another important feature to consider, is that such methods are stochastic methods, meaning that applying the algorithm to the same instance of the problem may result in different solutions. To such category, the following algorithms can be successfully applied:

- **Genetic Algorithms (GAs)** [9, 10]: a set of candidate solutions representing “chromosomes” are subject to the Darwinian evolution, i.e., the best candidates in the current population (the set of candidate solutions) are fused together (crossover) and eventually mutated in order to generate new candidates. Each potential solution is evaluated w.r.t. a fitness function which is strictly related to the OF. Iterating this process, the algorithm can end with a given (near-)optimal solution. GAs are generally limited by a low level of abstraction that can be used to describe the structure of the solutions, as well as by the set of operations (crossover, selection, mutation) that must be applied to the solutions.

  - To overcome some of the limitations found in GAs, different forms of evolutionary algorithms based on the natural-selection idea have been proposed. To this category, Evolution Strategies (ES) can provide good results [11, 12].

- **Ant Colony Optimization (ACO)** [13, 14]: this class of evolutionary systems, try to mimicking the behavior of ants in a colony. Ants, generally are demanded to find food for the colony, and can interact each other by emitting pheromones that can attract other ants. So, the higher is the pheromone the higher is the probability of having found a good food source. The idea is to randomly sampling the search space of the problem, and selecting over the time the area that is much promising for finding a good candidate solution. This can be achieved by forcing the other candidate solutions (also representing ant agents exploring the search space of the problem) to move and sampling such areas. ACO algorithms have been demonstrated to provide better solutions (also w.r.t. GAs) for problems where it is required to traverse a graph.

- **Particle Swarm Optimization (PSO)** [15, 16, 17, 18]: this class is similar to that of ACO, however PSO have been proposed has a general way for the optimization of problems with continuous variables (i.e., the solution is expressed as a tuple of real numbers). By evolving over the time the position of particles (agents sampling the search space), which is interpreted as a probability of...
assigning a given task to a certain server, it is possible to successfully apply such strategy to the problem stated in the previous section. Among the advantages of PSOs, the high quality and robustness of the solutions are the most valuable. In particular robustness means that the strategy tends (it is still a stochastic method) to compute the same solution given the same problem instance.

- Other solutions that can be applied to find a good solution [19, 20, 21]: constraint satisfaction problem (CSP), control theory, and game theory.

4.2 INTEGRATION OF THE HEURISTIC

This section describes how a PSO-based heuristic can be integrated into the resource allocation module described in section 2, as well as it provides a possible implementation for such dynamic scheduler. As future investigation work, we will evaluate other potential heuristics in order to integrate the most effective one.

PSO is a metaheuristic developed by Kennedy and Eberhart in 1995 to optimise multi-modal continuous problems. PSO is a population-based stochastic optimisation approach, where a group of independent solutions are used to sample the search space and discover the optimal solution. In PSO, a group of particles are evolved over time, by moving their position into a multi-variable search space. Passing from one position in a given instant of time to another position is made by taking into account the velocity of the particles. The particles' velocity and their positions are taken care by two components, which are described as two factors incorporating a form of distributed intelligence:

- **Cognitive factor**: encodes the information regarding the history of the best position assumed by the particles at the time $t$.
- **Social factor**: encodes the information relating to the history of the best position assumed by the neighbourhood of the particle at the time $t$.

These two factors are used to adapt the velocity of the particles in such a way it can steer the position towards the optimal solution. In PSO, there are no operators devoted to combining solutions belonging to the same population. The social factor allows to incorporate the knowledge collected by other particles. The topology of the neighbourhood influences the behaviour of the heuristic, although the entire set of particles is used as the neighbourhood (i.e., **lattice model**). The lattice model also has the advantage of keeping the number of operations used to determine the absolute best position low. The following equation shows the general rule used to update the velocity of the particle $i$ at time $t$:

$$v_i^{t+1} = \omega v_i^t + \varphi_1 r_1 (B_i^t - X_i^t) + \varphi_2 r_2 (\hat{B}^t - X_i^t)$$

In such equation, $\omega$ parameter is called **inertia factor**, and it is used to determine the fraction of the current velocity to use (i.e., it determines how fast we want to move the particle to the next position, compared to the current velocity). $B_i^t$ and $\hat{B}^t$ are respectively the best position assumed by the particle and by the whole swarm at the time $t$. $\varphi_1$ and $\varphi_2$ are parameters that greatly influence the algorithm convergence and are kept constant (several works demonstrated empirically that setting $\varphi_1 = \varphi_2 = 2$ provides the best trade-off between probability of convergence and algorithm efficiency). Finally $r_1$ and $r_2$ are two stochastic variable with a uniform distribution $U(0,1)$. The following equation shows how the position of each particle is updated:

$$X_i^{t+1} = X_i^t + \chi v_i^{t+1}$$

The parameter $\chi$ is called **constriction factor**. It can be used to adapt the final velocity in such a way the change to the position of the particle is small enough not to compromise the overall adaptation given by the cognitive and social factors.

In such adaptive optimisation heuristics, an initial set of solutions is randomly generated by the algorithm. In the case of PSO, solutions represent the particle's position in the search space and can be interpreted (eventually as the probability of assigning the task) as the assignment of a given task to a certain host server. Figure 6 shows the main steps performed by the evolutionary algorithm, based on the PSO model to compute the optimal allocation of the computing resources.
The algorithm starts by creating a random population (lines 3–5), i.e., a set of arrays, each representing a candidate solution. Since tasks and computing resources are represented by loads/offers regarding different internal resources (CPU, memory, network, storage, etc.), the array components (dimensions) are used to represent such resources. Dimensions correspond to the cardinality of the search space. For the specific problem, the size of the solution (i.e., the number of components of each solution array) can be equal to the number of tasks to assign. In line 6, the best ever solution is initialized to the position of the first candidate solution, while in lines 7–13 the the best ever solution (best_solution) and the best solution in the current population (best_particle) are set. In line 14–16 the initial velocity of each candidate solution (particle) is randomly generated. Lines 17–34 forms the main algorithm loop: till the maximum number of iteration is not reached, the population is evolved. Specifically, for each dimension two random numbers (r1 and r2) are pick, then the velocity of the particle is updated (i.e., for each dimension, the value of the velocity array is updated)—line 22. Once the new velocity is obtained, it is used to updated position of the particle (line 24). Based on the new position, the best solution in the current population and the best ever solution are updated accordingly to the value of the objective function OF and the position of the particle.

4.3 CONTAINER MIGRATION

There are several popular implementations of containers for Linux operating system, for instance, Docker, LXC, and runc. Those container implementations rely on CRIU tool for checkpoint, restore and migration of the containers.

CRIU is an acronym defined as "Checkpoint-Restore in Userspace". Although CRIU heavily relies on advanced features found in the Linux kernel, it does not require any modifications for the kernel itself and it is able to perform checkpoint and restore operations entirely in userspace.
At the basic level, the CRIU tool allows freezing a running application and checkpointing it as a collection of files. These files can be afterwards used for restoring the application and continue running it exactly from the point where it was frozen. This basic checkpoint-restore functionality allows application live migration, application snapshots, remote application analysis and remote debugging.

Any flavor of Linux containers can be abstracted as a process tree along with additional properties required for process isolation and fine-grained resource management. These processes may have access to various virtual and pseudo devices, such as veth or macvtap for networking or tty and pts for standard input and output.

CRIU is capable to snapshot the state of the entire process tree as well as the state of the virtual and pseudo devices the processes in the process tree are using. In addition, the properties required for process isolation and fine-grained resource management are saved and become an integral part of the container state snapshot.

### 4.3.1 Post-copy vs Pre-copy

Here we summarise the comparison between the two possible approaches for migrating containers between hosts: the post-copy approach, and the pre-copy approach. A more detailed comparison is available in deliverable D5.3. Pre-copy approach leverages on one or more round of memory pre-copy before freezing the container for the purpose of migration. In that case, it is possible to reduce the application downtime for the application. The main drawback of such approach is that it incurs in a too large overhead if the application presents a rapid change of the memory working set. Specifically, for such applications, the amount of modified memory will always be higher than the desired threshold and therefore the iterative pre-copy algorithm will never converge. On the other hand, the post-copy approach can be used. In this approach, the memory dump is not created and memory contents is transferred after the application is resumed on the destination node. There are several advantages for his second approach. First, it can guarantee the convergence (thus it allows to migrate containers) also in the case of rapidly changing memory working sets. Second, it requires less network bandwidth. Given this premise, in the following the way we implemented post-copy migration is provided.

### 4.3.2 Post-copy Migration Implementation

**Userfaultfd**

The fundamental requirement for post-copy migration is the ability of the software controlling the migration to intercept page faults generated by the application that is being migrated. In Linux operating system this ability is provided by a mechanism called userfaultfd [22].

This mechanism was implemented initially as a part of ORBIT EU project to enable post-copy migration of virtual machines.

During the OPERA project we are extending the userfaultfd mechanism with additional features required for implementation of post-copy container migration.

The userfaultfd mechanism is designed to allow a thread in a multi-threaded program to perform user-space paging for the other threads in the process. When a page fault occurs for one of the memory regions registered to the userfaultfd object, the faulting thread is put to sleep and an event is generated that can be read via the userfaultfd file descriptor. The fault-handling thread reads events from this file descriptor and services them using the operations provided by the userfaultfd mechanism. These operation include ability to copy a contiguous memory chunk into the faulting thread address space or zero out continuous memory range in the faulting thread address space.

This functionality is sufficient for post-copy migration of virtual machines with KVM hypervisor. In KVM virtualization model, each virtual CPU is a thread in a multi-threaded program (QEMU) running as the user-space part of the KVM hypervisor. Therefore, memory accesses of the guest are seen by the hypervisor as memory accesses of a thread in a multi-threaded program. The QEMU program is extended with additional thread that manages and monitors userfaultfd objects and implements user-space paging for the threads representing virtual CPUs.

Another limitation of prior userfaultfd implementation is the kinds of memory mappings it was able to support. In its initial implementation, userfaultfd allowed working only with memory regions that were
mapped by the application as private and anonymous. In other words, the mappings that are not backed by any file (including virtual filesystems) and visible only to the process that mapped them.

The container migration tool CRIU does not share memory address space with the processes running inside the container, and, hence, existing functionality of userfaultfd is not sufficient for implementation of post-copy container migration.

If a process that is being restored changes its virtual memory layout using system calls available in the Linux operating system, such as munmap(), madvise(MADV_DONTNEED) and mremap(), CRIU has no way to determine that these changes occurred and will cause a memory corruption.

Additional system call that needs special treatment is fork() system call. When the faulting process creates a child process using the fork() system call, the child is created with missing memory regions and there is no way the operating system can properly map these regions.

As a part of OPERA project we implement so-called "non-cooperative" extensions to the userfaultfd mechanism [23, 24]. These extensions allow precise tracking of the changes in the process virtual memory layout and creation of the child processes.

The non-cooperative extensions to userfaultfd include the following components:

- Modifications to Linux system calls that may modify the virtual memory layout, so that each invocation of such system call will cause a callback into the userfaultfd;
- Modifications to the fork() system call, so that each invocation of this system call will cause a callback into the userfaultfd;
- The implementation of the above callbacks in the userfaultfd core mechanism;
- Addition of ability to generate non page-fault events to the userfaultfd mechanism;
- Generation of events upon invocation of modified system calls by the faulting process.

Figure 7 illustrates differences in page fault handling with userfaultfd in cooperative and non-cooperative modes:

![Figure 7 - Page fault handling with cooperative and non-cooperative userfaultfd.](image)

Except the non-cooperative extensions to userfaultfd mechanism, during the OPERA project we've also added ability to use userfaultfd with shared memory regions. This includes memory regions backed by files residing on tmpfs filesystems, memory areas created with System V complied shm calls, areas created using memfd_create() system call and areas explicitly mapped as shared using mmap() system call.
CRIU

CRIU implements the container migration using checkpoint-copy-restore scheme. On the source node, the applications inside the container are frozen, their complete state is saved into image files. The image files are then transferred to the destination node. After the transfer is complete, the applications are restored on the destination node, along with additional properties defining the encapsulating container. The application down-time can be reduced with one or more rounds of iterative memory pre-copy.

During the OPERA project we extend CRIU project with ability to perform post-copy memory migration, or, in CRIU terminology, lazy migration [25, 26]. Our implementation utilizes userfaultfd mechanism to track memory accesses of the restored applications and to resolve the page faults caused by those memory accesses.

Outline of lazy migration with CRIU is:

- Start dump operation on the source node, use appropriate options to ensure that the memory of the checkpointed processes is not written into the images by rather kept in the main memory;
- Transfer images containing minimal checkpoint to the destination node;
- On the destination node, start special daemon called lazy-pages, that will be responsible for the user-space paging of the application that is being restored;
- On the destination node, start the restore operation, use appropriate options to ensure that the application is restored without its memory contents;
- When the restored application accesses missing page, the userfaultfd notifies the lazy-pages daemon about the page fault;
- The lazy-pages daemon requests the page at the faulting address from the source node;
- After the page contents is transferred to the source node, the lazy-pages daemon injects this page into the restored process address space;
- The restored process resumes its execution;
- The lazy-pages daemon fetches the memory pages that were not yet accessed by the restored process in the background;
- Once all the memory is transferred, the lazy-pages daemon exits and the migration completes.

4.4 FINAL REMARKS

The dynamic optimization procedure has a good potential in reducing the overall power consumption of the data center infrastructure. In fact, only at this level seems to be reasonable to reducing the number of active servers. The optimization strategy aims at finding an allocation scheduling able to reduce at the same time the number of used machines and the overall power consumption, by packing as much as possible the tasks (server consolidation). Since, the large quantity of power consumption in the servers derives from the idle fraction, switching off unused machine would provide a great benefit. In this context, container migration (also possible between heterogeneous resources), represents a worth tool. Future work will be oriented in studying and simulating such strategies (and also to improve static allocation).
This chapter provides some initial experimental results to show the current structure and organization of the ECRAE module and to demonstrate the capability of the resources allocation strategy under development, and the CRIU system. Current version of the ECRAE module (it is written using Python) does not integrate the dynamic scheduling approach that is under development, as well as does not provide integration with the CRIU system for Linux container migration.

5.1 TASKS DEPLOYMENT EXAMPLE

The aim of this section is to provide some initial results by simulating the allocation of cloud applications. Applications are simulated by creating a queue of tasks to be allocated on the data center infrastructure. Similarly, the data center infrastructure is described by the set of available nodes. Each node is also associated to an affinity tag, in order to allow the ECRAE to restrict the task allocation to the nodes that satisfy the affinity requirement. Specifically, by showing how each application increase the load of different available nodes, the purpose of the simulations is to show how the selected nodes change over the time (i.e., once the corresponding load -- CPU and Memory usage -- increases). This results in showing how the increased load on specific nodes lead the ECRAE to steer decision toward different ones, although trying to reduce the global power (energy) consumption.

Figure 8 - Example of a set of tasks (cloud applications’ components): each task provides the $\alpha$ value, the CPU and memory load, as well as the associated affinity tag.

Figure 8 shows the structure of the task list to allocate on the data center infrastructure. Each task is characterized by the parameter $\alpha$ that is used to weight the CPU load vs. the memory load, by the task relative load (CPU and memory load are normalized so that they remain comprised between 0 and 1; this is a common way of representing loads in cloud scheduling policies). Finally, the affinity tag is also provided.

On the other hand, figure 9 shows an example of the structure of the list of available machines in the data center. Each node is identified by a unique identifier in the data center (node_id), and by the affinity tag associated to that node. Then, two columns allow to express the current load of the machine in terms of CPU and memory load. As in the case of task, CPU and memory load are normalized. A simulation is run by issuing “simulate” command on the ECRAE client interface. Such command enables the system to read the list of task to assign, and for each task generate the correct allocation. Nodes are ranked according to the model discussed in section 2.1.2 (the score assigned to each node takes also into consideration the power drawn by the nodes in idle state by assuming a consumption equals to 65% of the whole power consumption).
Figure 9 - Example of the set of data center nodes available, each with its initial load.

Figure 10 shows the process of mapping tasks on the available resources. Task are analyzed one at a time to correctly take a decision.

Figure 10 - Example of mapping input tasks on the available resources.
Finally, figure 11 shows the results of the task mapping (assuming that only the greedy strategy has been applied). Each of the task is mapped on a corresponding node, according to the affinity tag expressed in the task input queue. As the reader can see in the example, the first five tasks are regularly mapped on the best node found, as well as tasks 8 and 10. Conversely, for tasks 6, 7 and 9 is not possible to select an optimal node (all the node evaluated resulted to be overloaded, thus the strategy try to find a less consuming node). Since, all the nodes resulted to be overloaded, the strategy selects the first in the list of available nodes (the one that provided the lower score anyway).

5.2 MIGRATION EXAMPLE

The following example presents migration of `memcached` from node `src` to node `dst`. The figure 12 illustrates a command sequence used to start the `memcached` service and verify that it is functional on the source (src) node. The last command initiates the migration by starting the `criu dump` with the options required for post-copy memory migration.

The figure 13 shows the sequence of commands used to complete the `memcached` migration on the destination (dst) node.
And, finally, the figure 14 represents verification that *memcached* is working on the destination node.
6 CONCLUSIONS

This document presents a summary of the work carried out in work package 5 (WP5) - task T5.4. The main objective of such research activity is to define a software system (referred to ECRAE–Efficient Cloud Resources Allocation Engine–in the document) that is responsible to efficiently allocate data center resources to the cloud application components. Here, resources are considered as “computing resources”, thus they are represented by the set of server machines available to run the software components.

The ECRAE uses a greedy algorithm to initially allocate resources, depending on the current status of the host servers in the data center. In particular, it tries to allocate the application component (the cloud application is described through a standard description format called TOSCA; generally, cloud applications are composed of a set of separated components –database, web interface, etc.– interacting each other) to the server that has the best score in terms of power consumption. Specifically, the servers are ranked depending on their current load and power consumption and the less loaded (i.e., the one that provides the lower power consumption) is selected. Since, servers in a (near-)idle state can consume up to 70% it is required to have the minimum number of active machines to achieve good levels of energy efficiency. To this end, the ECRAE periodically reschedules the application components on the server machines, with the aim of reducing the number of active machines and, at the same time, reducing the overall power consumption (energy consumption depends on the duration of the activity and it is proportional to the power consumption). In this context, migration of traditional virtual machines, as well as of more efficient Linux containers is also discussed. A CRIU-based mechanism to allow moving containers also between nodes exposing different architectural features (e.g., the ISA is different, the amount of memory, processor family, etc.).

As part of the ‘lesson learned’, we designed and implemented a software module that allows us to study how different strategies for allocating resources behave. We found that more potential in reducing overall power consumption (and thus energy consumed by the data center) can be obtained by globally optimizing the task allocation. This can be accomplished by an optimization algorithm, whom target is the reduction of the number of active machine and the overall power consumption. However, we also found that improving the way ECRAE statically allocates the tasks (Phase-1, see section 2.1) may reduce the number of subsequent migrations of virtual machines and containers.

Future activities related to T5.4 will be focused on the integration of ECRAE into the OpenStack environment, as well as the integration of the CRIU system in the chain to enable use of more efficient Linux containers. We will also evaluate more accurately the policies put in place by the ECRAE to allocate resources, eventually considering alternatives.
7 REFERENCES


[24] “Userfaultfd documentation update”, https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=5a02026d390ea1bb0c16a0e214e45613a3e3d885
