Large-scale EXecution for Industry & Society

Deliverable D4.4

Definition of workload management policies in federated Cloud/HPC environments

Co-funded by the Horizon 2020 Framework Programme of the European Union
Grant Agreement Number 825532
ICT-11-2018-2019 (IA - Innovation Action)

DELIVERABLE ID | TITLE	D4.4 | Definition of Workload Management Policies in Federated Cloud/HPC Environments
RESPONSIBLE AUTHOR	Martin Golasowski (IT4I)
WORKPACKAGE ID | TITLE	WP4 | Orchestration and Secure Cloud/HPC Services Provisioning
WORKPACKAGE LEADER	Alberto Scionti (LINKS)
DATE OF DELIVERY (CONTRACTUAL)	31/03/2020 (M15)
DATE OF DELIVERY (SUBMITTED)	30/06/2020 (M18)
VERSION | STATUS	V1.4 | Final
TYPE OF DELIVERABLE	R (Report)
DISSEMINATION LEVEL	PU (Public)
AUTHORS (PARTNER)	IT4I, LRZ, LINKS, ATOS
INTERNAL REVIEW	Stephane Louise (CEA), Frederic Donnat (O24), Stanislav Böhm (IT4I)

Project Coordinator: Dr. Jan Martinovič – IT4Innovations, VSB – Technical University of Ostrava
E-mail: jan.martinovic@vsb.cz, **Phone:** +420 597 329 598, **Web:** https://lexis-project.eu

DOCUMENT VERSION

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	Table of Contents, outline	21/01/2020	Martin Golasowski (IT4I)
0.2	Detailed table of contents	04/02/2020	Martin Golasowski (IT4I)
0.3	Filled in sections required from LRZ	27/02/2020	LRZ (all)
0.3.1	Reviewed and updated sections requiring proof reading from Atos (Sections 3.1 & 3.2)	28/02/2020	Marc Levrier (Atos)
0.4	Added state of the art on scheduling policies, description of Orchestration Service, description of dynamic resource selection mechanism and updated Glossary.	06/04/2020	Alberto Scionti, Giacomo Vitali, YuanYuan Li (LINKS)
0.4.1	Section 2 review	26/04/2020	Alberto Scionti, Giacomo Vitali, YuanYuan Li (LINKS)
0.5	Document restructuring, rewriting, and correct wording	30/04/2020	Martin Golasowski (IT4I)
0.6	Revising and reorganization of Subsection 4.3	19/05/2020	Alberto Scionti, Giacomo Vitali (LINKS)
1.0	Formal check, added summary, preparation for internal review	21/05/2020	Martin Golasowski (IT4I)
1.1	Internal review version	22/05/2020	Martin Golasowski (IT4I)
1.2	Addressing reviewers´ comments	06/06/2020	Alberto Scionti (LINKS)
1.3	Preparation for final submission	26/06/2020	Martin Golasowski (IT4I)
1.4	Final check and minor changes	29/06/2020	Kateřina Slaninová (IT4I)

GLOSSARY

ACRONYM	DESCRIPTION
AAI	LEXIS Authentication and authorization interface
DDI	LEXIS Distributed Data Interface
HDD	Hard Disc Drive
HEAPPE	High-End Application Execution Middleware *former* HPC as a Service Middleware
HPC	High Performance Computing
MOC	Model of Computation
NVME-OF	Non-volatile Memory over Fabric communication protocol
SBB	Smart Burst Buffer
SBF	Smart Bunch of Flash
SSD	Solid State Drive

TABLE OF PARTNERS

ACRONYM	PARTNER
Avio Aero	GE AVIO SRL
AWI	ALFRED WEGENER INSTITUT HELMHOLTZ ZENTRUM FUR POLAR UND MEERESFORSCHUNG
Atos	BULL SAS
BLABS	BAYNCORE LABS LIMITED
CEA	COMMISSARIAT A L ENERGIE ATOMIQUE ET AUX ENERGIES ALTERNATIVES
CIMA	Centro Internazionale in Monitoraggio Ambientale - Fondazione CIMA
CYC	CYCLOPS LABS GMBH
ECMWF	EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS
GFZ	HELMHOLTZ ZENTRUM POTSDAM DEUTSCHESGEOFORSCHUNGSZENTRUM GFZ
IT4I	VYSOKA SKOLA BANSKA - TECHNICKA UNIVERZITA OSTRAVA / IT4Innovations National Supercomputing Centre
ITHACA	ASSOCIAZIONE ITHACA
LINKS	FONDAZIONE LINKS / ISTITUTO SUPERIORE MARIO BOELLA ISMB
LRZ	BAYERISCHE AKADEMIE DER WISSENSCHAFTEN / Leibniz Rechenzentrum der BAdW
NUM	NUMTECH
O24	OUTPOST 24 FRANCE
TESEO	TESEO SPA TECNOLOGIE E SISTEMI ELETTRONICI ED OTTICI

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

Dynamic orchestration of HPC and Cloud workloads is one of the main goals of the LEXIS project. In order to successfully design the workload placement policies, metrics, constraints and requirements coming from infrastructure and use cases must be collected. Initial proposal of the dynamic placement policies and usage of the collected requirements and metrics is provided in this document. The policies will be further extended and formalized during the project with experience gained from the pilot use cases and Open Call applicants.

Position of the deliverable in the whole project context

This report defines metrics and requirements coming from the infrastructure (DDI, Cloud and HPC and the platform users — pilot use cases (WP5-WP7)). It is part of the Task 4.4 Integration of the Overall HPC/Cloud Orchestration System. This report collects important requirements and constraints needed to design and implement the final orchestration system which will be described in Deliverable D4.6 [1], due in M30.

Description of the deliverable

The document is divided into three main sections, the Section 2 focuses on collection of metrics provided by the executed workflows provided by WP5-WP7. Section 3 describes requirements and constraints imposed by the underlying computing and storage infrastructure. Content of these sections will allow to precisely formulate requirements on workflows which would like to use the LEXIS platform and on infrastructure providers which would like to allow the platform to execute the workflows on their infrastructure. Besides that, the sections will provide integral input to the design of the dynamic orchestrator, which is described in Section 4. Section 5 then provides common glossary of terms using to describe the LEXIS orchestrator context.

1 INTRODUCTION

LEXIS project will provide an execution platform for running complex workflows on federated HPC and Cloud resources. The effective orchestration of application workflows represents a key point of the LEXIS platform. Among the others, *workload management policies* (i.e., the mechanisms used to make decisions on which resources to allocate workflow tasks) are at the basis of any effective orchestration solution, since they allow to select the set of execution resources that better fulfil the applications and infrastructural requirements and constraints.

LEXIS orchestrator has been designed to provide, in its final form, an effective mechanism to select the computational and storage resources to be used for executing workflow tasks: resources will be statically specified in the workflow or will be dynamically selected by using evaluation criteria that will combine both the information returned by the monitoring system and also application/user inputs. Moreover, workload management policies will take advantage of the diversity of resources available at each computational site (IT4I and LRZ), which include Cloud, HPC, Burst Buffer types.

To this end, YSTIA orchestration engine, along with the Alien4Cloud frontend (both developed by Atos), have been extended in order to be able to use HPC resources, other than Cloud ones. Through a plugin for specialized middleware (HEAppE), the YSTIA orchestration engine can launch jobs on HPC cluster resources. As part of the LEXIS workload management policies, YSTIA solution implements a mechanism for statically define the resources to use. Final version of the LEXIS platform will also implement a mechanism for dynamically selecting resources at the workflow run-time, based on a service which will provide such information to the YSTIA orchestrator.

In the following sections, the requirements and constraints (collected both from pilot use cases and infrastructure), the solutions devised to effectively execute the LEXIS workflows and consequently the specific mechanisms for managing the allocation of the workload on the available infrastructural resources will be presented. Being part of the service-based LEXIS platform, the LEXIS Orchestration Service and its main architectural components are presented in this document. The remainder of this document is organized as follows. Pilot use cases and infrastructural requirements and constraints are discussed in Section 2 and 3. Section 4 contains the details on the definition of the *workload management policies* which are defined in terms of static and dynamic resource allocation policies. Lastly, Section 5 provides the definition for the most used terms in the scope of the LEXIS project.

2 RESOURCE ALLOCATION REQUIREMENTS IN PILOT APPLICATIONS

The analysis of the LEXIS pilot use cases allows to identify requirements from which the workload management policy definition should be derived. The following subsections briefly summarize the Pilot use cases and, for each of them, highlight main requirements. The way these specific requirements can be fulfilled is given in Section 4.

2.1 WP5 (AERONAUTICS)

This pilot includes two large-scale Aeronautics test beds, one referring to a Turbomachinery application and the other one concerning a Rotating Parts case study. Being CPU-demanding, data intensive and time-consuming, both the case studies can be informally described as typical HPC use cases.

These rely on sophisticated Computer-Aided Engineering (CAE) tools that are adopted to examine the behaviour of complex flows in critical components of an aeronautical engine through Computational Fluid Dynamics (CFD) simulations running on HPC systems. More specifically, as in any standard CAE analysis, such CFD simulations include the following three phases [2, 3]:

- Pre-processing or uploading of input data,
- Simulation solver execution on HPC resources,
- Post-processing and visualisation of simulation result.

The first phase prepares or simply uploads the input data for the main simulation task. The data must be staged on the HPC cluster storage by the DDI to avoid unnecessary data transfers. Staging area on the DDI must be large enough to handle the data used by the application workflows provided by this pilot. The second phase, in the end, is expected to have a run-time of several weeks for the Turbomachinery use case, and several hours or days for the Rotating parts one depending on the investigated mesh resolution.

From the analysis of these use cases the following requirements emerge. The long-lasting simulation phase demands a checkpointing mechanism, which must be implemented at the application level and the orchestrator must support it by providing capabilities to handle data copy and repeated execution after a job failure.

Data locality also ensures optimal I/O performance when executing the post-processing and visualisation tasks which will be run on the HPC cluster as well, using large-memory GPU-equipped graphic node. Worth to remark here, is the fact that the target locations for executing the tasks associated to these three phases can be already determined at the time of the execution of the workflow (static allocation policy – see Section 4). However, locations that are more suitable for running workflow tasks can be found at run-time by leveraging on monitoring information. For instance, this can be true for the post-processing tasks performed in Cloud.

2.2 WP6 (EARTHQUAKE AND TSUNAMI)

Workflows provided by this pilot are focused on simulation of possible tsunami occurrence after an earthquake and a damage estimation. It uses geospatial database (PostGIS) with processed OpenBuildingMap data, which are continuously updated. Execution of the tsunami simulation is triggered by an earthquake event or by user input (what-if simulation). The simulation must provide results within a limited timeframe since the results are essential for disaster management. Redundant execution of the workflow must be supported to ensure high availability of the results (the same simulation is executed in multiple centres at once). A lower precision simulation can be executed in this mode to ensure availability of the results in short time, while a higher simulation that takes longer time can be executed in parallel to support the disaster management process.

Therefore, main requirements for the orchestrator can be identified in the support for event driven and periodic execution. Support for prioritized execution to meet the imposed deadlines and simultaneous execution on multiple resources to ensure high availability is also required [4, 5].

2.3 WP7 (WEATHER AND CLIMATE)

This pilot use case focuses on large-scale climate simulations using different models. Simulation outputs are thus used to feed different models targeting various applications, including fire risk prediction, hydrological models, and air quality analysis. Although targeting different possible applications, most of the processing stages contained in this use case workflows are in common (*e.g.*, WRF simulations), and large data sets are produced as well as consumed as an input. Results of the simulations are also post-processed and visualised. The combination of highly computational demanding workflow steps and less demanding ones makes the case for the use of both HPC and Cloud infrastructural resources [6].

As the first requirement, the orchestrator must ensure data locality and minimize the needs of data transfer. As a consequence, Burst Buffer storage acceleration will most likely prove as essential in this pilot. The simulations are also long-term running, therefore meet the same requirement on checkpointing and resiliency support as requested by WP5. Finally, WP7 use cases, along with WP6 use case, are those where urgent computing requirements are the most prominent, as described in Section 4.4.

3 INFRASTRUCTURE CONSTRAINTS AND REQUIREMENTS

This section describes requirements and constraints coming from the infrastructure that the orchestrator will address. Metrics that can be provided by the centres to drive the dynamic orchestration are also listed here.

3.1 CLOUD

The cloud environments at IT4I and LRZ are implemented by OpenStack and provide on-demand virtual machines, tenant networks and storage. IT4I operates experimental cloud which is dedicated purely for LEXIS and can be reconfigured as requested, while LRZ operates production cloud which has a strict access and usage policy.

3.1.1 Constraints

- Tasks intended to run on the cloud side may depend on results produced by other tasks running on the HPC par and the other way around.
- The technical capabilities or inabilities of the cloud environments may differ in each HPC service provider. The application workflow modelling in the orchestrator will have to take these into account, while masking their complexity to the end user of the platform.
- Orchestrator Notification functionality — for example monitoring of the workflow progress, low utilization of VMs in the Cloud part, etc.
- The orchestrator must support the OpenStack API.

3.1.2 Orchestration metrics

- Resources availability
  - Available resources for Cloud
  - Cloud resources allocation within an HPC computational project (cloud credits)
- System availability
  - System status — operating, maintenance, down
  - Planned maintenance windows
- Availability of resources
  - Number of vCPUs, RAM
  - Storage types available (SSDs, Burst Buffer, HDDs) and size
  - Networking (public IPs available, tenant networks)
- Requested allocation
  - Provided by: Cloud API

3.2 HPC

HPC systems are characteristic for their batch execution mode. The systems are shared by many users and the applications are executed on multiple bare-metal nodes with exclusive access. The applications are executed in jobs which are scheduled for example by PBS Pro, Slurm, Torque and similar solutions.

Jobs are usually scheduled in multiple queues, each of which can correspond to a different type of resource (CPU nodes, GPU nodes, visualisation nodes, etc.). Each queue can have also a different priority. HPC centres implement their own allocation policy, which can include queue priority, amount of already consumed core hours in the current HPC computational project and other criteria.

3.2.1 Constraints

- Batch execution
  - Multiple queues with different priorities and restrictions (nodes per job, resource type, etc.)
  - Scheduler interface available only through SSH and command line interface
  - The way jobs are scheduled by batch schedulers on HPC resources may differ at each HPC centre (consumed resources in the HPC computational project, allocation type, etc.)
- Normalized core hours
  - Consumption of core hours in single HPC centre on different clusters is multiplied by a certain factor; for example — factor 0.9 is set for older system, 1.0 for current production and 1.2 for new experimental one
  - Core hours in IT4I are normalized
  - LRZ does not implement normalized core hours – can be provided by the LEXIS platform
- Resource usage monitoring to avoid resource wasting
  - Under-utilization of CPU
  - Process hanged or frozen
  - Disproportionate I/O time
- Allow exclusive selection of an HPC centre
  - Only certain centres may be used to allocate tasks in a given workflow
  - Data migration constraints (sensitivity, security, migration efficiency)
  - Intellectual property constraints (patents)
- Cluster utilization balancing
  - Make sure that the resources are consumed fairly among different centres

3.2.2 Orchestration metrics

- HPC Computational project availability
  - Available resources (core hours) for HPC clusters
  - Provided by a dedicated approval system
- Systems availability
- Basic monitoring for each system connected to the LEXIS Platform (available, down, maintenance, etc.)
  - Planned maintenance dates
  - Provided by HPC Centre internal monitoring system
- Resource availability
  - Resource availability specific system (GPU, accelerators, storage, etc.)
  - Amount of available resources
  - Dedicated resources (static allocation, dedicated queue with high priority)
  - LRZ provides only standard dynamic batch scheduling mode on HPC clusters
  - Provided by HPC Centre internal monitoring system
- Requested allocation
  - Resources requested by the user/task (no. CPU cores, GPU nodes, etc.)
  - Provided by: HPC Scheduler

3.3 ACCELERATION

The LEXIS platform investigates FPGAs and Burst Buffer (BB) nodes as means to accelerate storage I/O and provide optimal data locality for applications executed on HPC and Cloud resources while avoiding unnecessary data movement.

3.3.1 Field Programmable Gate Array (FPGA) Accelerator

Traditionally, FPGAs are programmed using hardware description languages such as VHDL or Verilog. This approach still gives the best efficiency, but it is slow and prone to errors. Modern tools work with mixed C/C++/OpenCL code (HLS synthesis) and provide a software layer able to manage data transfers between the CPU main memory and the FPGA card, thus making algorithm acceleration easier. However, effort spent on porting the application to support the FPGA is still necessary.

Following the LEXIS Pilot Workflows requirements, the acceleration tasks under investigation are:

- Data compression/decompression, mainly focusing on accelerating the popular zlib library
- Data encryption, for IP security reason, focusing on OpenSSH supported algorithms
- Basic data or image manipulation (i.e. GRASS GIS tasks)

It is possible to execute a task on the BB node, where certain computation intensive parts of the code are offloaded on the FPGA card. This can be beneficial especially in processing of extensive data sets or high bandwidth data streams.

Application which uses the FPGA resources is tied to version of the SDK supported by the specific firmware flashed on the FPGA board itself. When the application is launched, the FPGA support driver loads the corresponding programming file on the FPGA. This version check is necessary because at power up the FPGA is programmed with the PCI express and SDRAM control logic, while the user accelerator is then loaded at runtime using the “partial reconfiguration” FPGA feature.

In addition to the plain OpenCL SDK runtime environment, Intel is also working on a more complete stack aimed at a datacentre distribution: this solution is known as Intel Acceleration Stack and is based on OPAE (Open Programmable Acceleration Engine). OPAE is a set of drivers, user-space libraries, and tools to discover, enumerate, share, query, access, manipulate, and reconfigure programmable accelerators; it is an open source solution freely available on GitHub [7]. OPAE user-space library contains a set of objects that can identify and reference FPGA accelerator resources; moreover, they provide a way to acquire, access and release the accelerators from the user applications, even if running inside a virtual machine.

The orchestrator must ensure that correct design is pre-loaded on the card and the environment is prepared for the application execution.

Intel Quartus software (used to build the final FPGA bitstream) can provide the following metrics:

- FPGA Utilization (as % of the internal resources, such as LUT, FF, BRAM, DSP),
- FPGA expected power requirement.

The FPGA runtime can provide these metrics:

- Performance stats (actual power as sensed by internal regulators, card, and FPGA temperature, QSFP status),
- Card properties (vendor, SDK version),
- Occupation status (available, down, in-use, etc.).

All these metrics collected from the FPGA cards become even more interesting from the orchestration standpoint, if we consider future improvements of the SDKs, which are expected to provide more abstraction and capability of sharing hardware resources among different applications and virtual machines (e.g., the Intel OPAE [7]).

3.3.2 Burst Buffer nodes

The burst buffer nodes are servers equipped with fast NVMe storage and connected to the Cloud and HPC infrastructure by a high bandwidth network. The Burst Buffer software provided by Atos then allows to dynamically allocate logical volumes on the NVMe drives and expose them to the Cloud and HPC on a per application or per workflow basis. Since this is one of the core innovations of the whole LEXIS platform, the LEXIS orchestrator must

work with the Burst Buffers to handle allocation and deallocation of the volumes and provide access to the DDI which then handles the data staging.

The Burst Buffer software provides two different modes of operation that are controlled by **parameters** given at launch time:

- **Smart Bunch of Flash (SBF) mode**: The burst buffer creates a persistent NVMe volume that is spread over all NVMe devices, then creates an XFS file system in it and exports it to one compute node using the NVMe-OF protocol. In this mode, the parameter is the size of the volume. Several volumes can be exported to different compute nodes. Each compute node will have its own dedicated volume. This mode can be also used to export such volume to the OpenStack volume management service Cinder.
- **Smart Burst Buffer (SBB) mode**: The burst buffer acts as a transparent file system cache associated to one job or a set of consecutive jobs using the persistent buffer feature. A dedicated software daemon is launched for each instance of the service. In this mode of operation, many mandatory and optional parameters are needed at launch time:
    - Number of cores or the CGROUP in which the daemon is launched,
    - RAM size used as L1 cache level (raw value, the actual data that can be cached is smaller due to the overhead of data structures),
    - NVMe disk space used as L2 cache level by SBB or exported volume by SBF (raw value, the actual data that can be cached is smaller due to the overhead of LVM and the file system).

When the Burst Buffer services are launched, run time metrics are also available in both modes of operations. Some metrics are common to both modes, while others are specific.

- **SBF mode**: no specific metrics in this mode.
- **SBB mode**: The SBB daemon provides through an administration command many internal metrics such as L1 Cache throughput, L2 Cache throughput, L1 cache IOPS, L2 cache IOPS: The value of these run time metrics could be roughly estimated according to the available hardware (e.g. NIC, NVMe disks, etc.), to the allocated resources for the SBB daemon (e.g. number of worker threads, etc.) and the load of the machine (e.g. number of SBB daemons running in parallel on the same node).
- **Common**: all standard InfiniBand and LVM2 tools can be used to get dynamic metrics.

3.4 DDI

The DDI is designed on top of the iRODS [8] / EUDAT-B2SAFE storage system, therefore main constraints by the DDI infrastructure will relate to characteristics of a typical iRODS system. The main potential bottlenecks are the storage systems used as backend, and the PostgreSQL based iCAT (metadata) database of iRODS.

The main objective of orchestration with respect to DDI is to minimize expensive data transfer operations, and to avoid out-of-space problems as well as overload of DDI systems. Therefore, we have the following principal optimisation objectives:

- Different parts of the workflow should be executed at one computing centre in the absence of other reasons (Cloud/HPC infrastructure full, etc.)
- Workflows should be executed at the same supercomputing centre as the data is located, unless the sum of the cost of the transfer plus the computation is smaller in other centres.

3.4.1 Constraints

- HPC parallel file system must be accessible to an iRODS client to enable data staging.
- Dedicated network with enough bandwidth must be available to both Cloud and HPC infrastructures.
- Backend file system for DDI must have enough space free.
- Orchestrator must consider limited I/O bandwidth available.
- Potential workflow regulation options (selected by the user): parts of the workflow must be computed in specific HPC centre (due to matching Supercomputing allocations or characteristics of Cloud Systems).
- Periodic transfers of intermediate results to another centre.

3.4.2 Orchestrator metrics

- Number of compute-centre switches within a workflow, with a low weighting factor (e.g. 0.1 or even 0.0) for switches with a concrete reason,
- Cost estimate for data transfer,
- Bandwidth measured between the different centres, and the different staging areas within the centres,
- Bytes transfer required for transfer between steps, including cost factor,
- Free space available in all tiers,
- System availability metrics,
- Ping check,
- Planned maintenance check,
- Access rights verification.

4 ORCHESTRATOR INTEGRATION

This section describes the LEXIS Orchestrator design. Differences between static and dynamic workload placement policies are described. The description takes into account requirements and constraints collected from the pilot use cases (Section 2) and infrastructure (Section 3). Initial formulation of the underlying optimization problem and its solution is provided along with correlation with the imposed requirements and constraints.

The main goal of the orchestrator is to execute the workflow task corresponding to the workflow specification on the most appropriate resources. We refer to these resources as the *location* where to execute the task. Thus, locations are representative of the resources made available on the HPC centres as HPC clusters or Cloud infrastructures.

The resource selection process is driven by the allocation policy implemented by the orchestrator, and does not affect the way the workload (i.e., jobs to be scheduled or VMs to instantiate) are managed at each location (e.g., once an HPC cluster is selected, the jobs associated to the allocated task are submitted to the batch scheduler, which will schedule their execution according to its own scheduling rules). The factors to be considered when designing the allocation policy are:

1. LEXIS platform relies on a geographically distributed infrastructure; thus, allocation of tasks should consider the cost of eventually moving data closer to the selected resource,
2. Cloud and HPC resources have different characteristics and constraints (i.e., OpenStack virtual instances need to be deployed before being used, while HPC jobs encounter a queuing time),
3. Applications can be executed only on a particular type of resource (GPU, FPGA, specific CPU architecture, etc.).

The YSTIA orchestrator developed by Atos (see Deliverable 4.2 [9]) is used at the core of the LEXIS Orchestrator component, which is described in detail in Section 4.2. The LEXIS Platform architecture is service based thus the Orchestrator will expose an API and will communicate with the other components of the platform. Section 4.3 describes the workload placement policy and Section 4.4 provides connection between the requirements specified in the previous sections and the proposed policy.

We also provide a current state of the art (SotA) in the domain of workload placement policies since it is a problem which has been already tackled in various related fields, such as Compute Grids (CGs). Geographically distributed resources are common property of the Compute Grids. Additional heterogeneity is introduced by various resources available in each HPC centre (multiple HPC clusters and cloud systems available with various architectures, storage systems and acceleration capabilities). We analysed several solutions that can inspire efficient solution of the resource selection problem with geographically distributed and heterogeneous resources.

4.1 State-of-the-art on distributed and Grid computing orchestration systems

Mechanisms for dynamic resource selection have been proposed both in Cloud and Grid computing domains. In the Cloud domain, many works proposed various approaches for selecting the resources to be used to run virtual machines, which are based on a defined optimality criterion.

Among different ways to approach the problem of resource allocation (*i.e.*, scheduling) in Cloud and Grid computing, a large effort has been spent in mapping the allocation problem as a combinatorial optimization problem, generally referred to as a (on-line) *bin-packing* problem [10, 11]. This latter is well-known to be NP-hard to solve; so, heuristics have been proposed. To mention few, in [12], the authors proposed to hybridize the cuckoo search heuristic with a gradient descent technique, to speed up the convergence of the algorithm towards the optimal solution. The algorithm was used to reduce the execution time of Cloud workloads. In [13] PSO heuristic was used to optimally schedule predicted workloads.

In [14], the authors surveyed several approaches for allocating resources in the Cloud domain. To mention just a few, Prodan et al. [15] proposed a bargaining-based resource scheduling technique (RST) in which market-based negotiation takes place between the resource manager and the scheduler using self-limitation and aggressiveness; Lyer et al. [16] proposed a method based on suggested pricing resource scheduling algorithms; in [17] the authors used a cost-based approach, where a simple First-Come-First-Served scheduling technique was applied. Other attempts used other type of heuristics, such as genetic algorithms [18] to allocate jobs using a cost-based evaluation function. Also, energy minimization has been tackled by solving an associated mixed linear programming problem [19]. The survey [14] completes the analysis of scheduling techniques with several approaches based on nature-inspired scheduling algorithms (*i.e.*, genetic algorithms, particle swarm optimization, ant colony optimization, etc.) [20, 21]. Additional scheduling approaches oriented to cover specific requirements that are common in weather forecast domain (and so in the related workflows) are reported in [22, 23]. Here, ways of considering urgent computing needs are discussed.

Apart from the used approach to scheduling, the main difference between Grid computing and Cloud domain stems in the distributed *vs* centralized nature of them. In Grid computing the assignment of resources is based on a *pull* mechanism instead of a *push* one. The push mechanism is common in the Cloud, where the orchestrator takes decision on the resources to use and then *pushes* the request to the selected nodes, where VMs are created and started. On the other hand, the pull mechanism is commonly used in Computing Grids (CG), where each resource signals its availability to the orchestrator and requests a new job to run. Examples of such pulling approach can be found in the SETI@Home and BOINC [24] world-scale projects. Among the various projects that use CG to provide computing resources, DIRAC [25] is one of the systems used to manage jobs in the CERN Computing Grid, specifically on the LHCb experiment. In the DIRAC platform [26, 27], resources that participate in the computing infrastructure have a local agent whose purpose is to interact with the centralized resource manager of the platform. Specifically, the centralized resource manager is responsible to collect input job requests, register each submitted job in a local database (containing job parameters and dynamic job status) and running the job in one of the available queues. Each queue maps jobs that can be run on specific resources. Periodically, optimizer functions are applied on the queues to reshuffle the jobs according to different criteria, and thus maintain a high throughput of the system. For example, in [26] an availability criterion is used. It states that total number of jobs assigned to a resource should not exceed a fraction ε of the total number of available CPUs

$$\frac{TotalQueueingJobs}{TotalCPUs} < \varepsilon$$

Whenever the execution resources monitored by a local agent become ready for the execution of a new job, the agent requests a job from the central management system. The management system then selects the job from the queue and sends the job to the requesting agent, along with a configuration package which is used to create the execution environment.

Compared to these two ways, LEXIS manages weakly coupled resources belonging to geographically distributed HPC centres. However, in contrast to traditional CGs, the orchestration of workload is done through a centralized component in the LEXIS architecture, which is similar to the Cloud approach. What LEXIS gets from the CG approaches is the use of a simple estimation of the most appropriate location where to execute new jobs or run VMs. This approach will be the focus of the next project period, where an algorithmic solution will be fully defined, implemented, and integrated with the remainder of the platform. The following sections will provide more details on the specific implemented approach.

4.2 ORCHESTRATOR ARCHITECTURE

In the co-design phase of the project has been decided that the architecture of the LEXIS will be service based. The Orchestration Service module (Figure 1), in its final implementation, will integrate all the features required to address user/application and infrastructure requirements and constraints previously described, as well as to implement a dynamic allocation policy. This section briefly describes the main service components with the

emphasis on the Business Logic which will implement the *dynamic* allocation policy (see Section 4.3). Overall architecture of the LEXIS platform is documented in Deliverables D2.2 [28] and D2.3 [29]. Details about the YSTIA (i.e., the software stack used as a base for the implementation of the LEXIS orchestrator) are in Deliverable D4.2 [9].

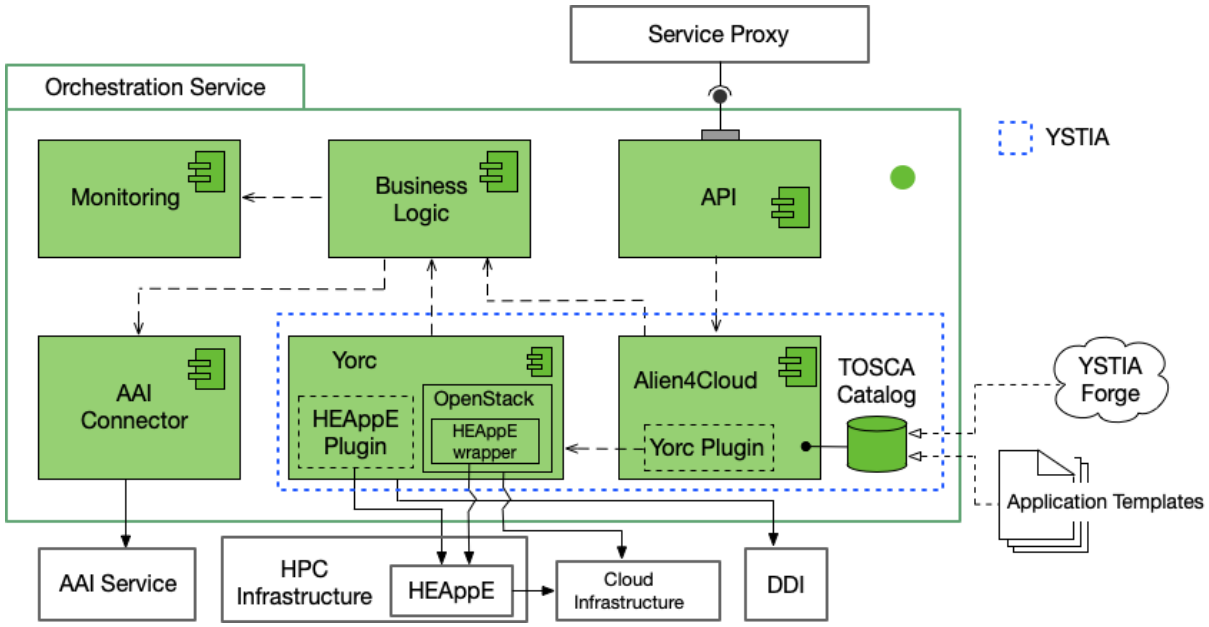

Figure 1 LEXIS Orchestration Service architecture (blue dashed box represents the modules belonging to YSTIA)

Core of the Orchestration Service module is the YSTIA software provided by Atos. It is composed of Yorc – YSTIA Orchestrator (*i.e.*, the actual orchestration engine which executes the workflows), Alien4Cloud – A4C (*i.e.*, the orchestrator engine frontend, which allows to model workflows) and YSTIA Forge (*i.e.*, a repository of TOSCA components used to define workflows, also containing public available ones). Alien4Cloud (A4C) provides an API and a GUI for describing YSTIA Application Templates using the TOSCA standard [30]. The templates are built from individual components which are stored in a catalogue. This catalogue can be extended by the platform developers on demand.

Yorc is an orchestration engine developed by Atos, which implements the static placement policy and executes the actual YSTIA Workflows on the selected resources. It uses: *i*) HEAppE (*i.e.,* a middleware developed by IT4I to securely getting access to the HPC cluster resources) via a special plugin to interact with the HPC clusters; *ii*) OpenStack API to interact with the Cloud; and *iii*) the API exposed by the LEXIS DDI to enable proper data migration between the different compute resources. Actually, A4C integrates a plugin to delegate the execution of YSTIA Application Templates on the underlying compute infrastructure to the Yorc engine. The same mechanism will be used to implement the *dynamic allocation policy:* a delegator component communicates with the Business Logic to obtain the location to use for the execution. Specifically, YSTIA workflows may contain delegators, which are a convenient mechanism to call an external function or service. Delegators can be used to require the resource selection for a given workflow task to the Business Logic. Thus, every time the workflow requires to dynamically selecting the location for executing a task, then business logic API is called.

The API component exposes the LEXIS Orchestration Service interface, which can be used by other services and modules, such as the LEXIS portal (developed in WP8 – see Deliverable D8.1 [31]). All API calls are authorized only upon successful authentication of the supplied token with the LEXIS AAI through the AAI Connector component.

The monitoring component provides orchestration metrics which are then used by the billing system and the monitoring system (Task 3.4) for auditing and operational purposes.

4.2.1 Business Logic component

This component, in its final version, will be implemented as a service with an HTTP API to integrate easily with the A4C delegators. It will connect to services providing measurable parameters of the infrastructural resources (*e.g.*, allocated virtual cores on the Cloud, number of enqueued jobs, etc.), as described in Sections 2 and 3, which will be used for dynamically selecting the most suitable set of resources to use, as described in Section 4.3.

Thus, both pilot applications use cases and infrastructural level requirements and constraints are taken into account by the LEXIS Orchestrator Service. To this end, such requirements and constraints have to find a mapping with the inner working mechanism of the orchestration service (*i.e.*, at the level of YSTIA modules or in the dynamic allocation strategy).

In the following, an example of the REST API for interacting with the Business Logic is provided:

- (POST) RequestLocations: it can be used to send a request for the best *N* locations where to run the next task/job; *N* can be passed as a parameter of the call.
- (GET) GetLocations: it is used to retrieve the list of the best-found *N* locations where to run the next job/task.
- (DELETE) DeleteRequest: this can be used to cancel the previously issued "RequestLocations" request.

4.3 WORKLOAD PLACEMENT POLICY

The workload placement policy defines a mechanism which assigns a workflow task to a particular compute resource while conforming to a given criterion. Within the context of LEXIS Orchestrator, we describe two types of placement policies – *static* and *dynamic.*

The static placement policy describes a mechanism where assignment of the resources is done *a priori,* and it is usually part of the workflow description. The assignment is done either by hand or automatically from a pre-defined list, not taking into account past or future state of the resource. In LEXIS Orchestrator, YSTIA already implements such policy, the placement of tasks is part of the TOSCA based description of the workflow.

On the other hand, dynamic placement policy assigns resource to a task while taking into account various inputs and conforming to a given criterion. While specification of a preferred *type* of a resource can be part of the workflow description, concrete assignment of a task to a particular compute resource (HPC Cluster, Cloud deployment, etc.) is done automatically, when a task is scheduled for the execution. It can be formulated as an optimization problem where criteria can be execution or data transfer time, overall cost, or other relevant ones. The dynamic placement policy also must take into account requirements and constrains defined in Sections 2 and 3. Together with application specific parameters and data provided by the infrastructure monitoring systems, various *metrics* can be formulated. For example:

- Size of the input data set, number of iterations, resolution, etc.,
- Utilization of an HPC cluster queues,
- Storage utilization (free space and current load),
- Amount of free resources in a Cloud instance,
- Upcoming planned downtimes.

Given the metrics example, optimization criteria can be formulated such as:

- Minimize execution time while taking into account necessary data transfer time and current utilization of the infrastructure (in correspondence to data placement policies defined in D4.3 [32]),
- Select a cloud infrastructure based on current free vCPUs, RAM or free space on a storage system,
- Ensure availability of the results by running same task on multiple resources.

There can be many more examples of metrics and criteria, their formalization is a subject of the upcoming work in the second half of the project. Refinement will be based mainly on knowledge obtained from deploying pilot use cases and workflows provided by the future OpenCall applicants.

Based on this analysis, the underlying optimization problem can be solved for example by greedy algorithm inspired by a similar approach used to optimize energy consumption of a data centre [33]. The metrics can be weighted, normalised, and combined to provide a single number which will describe suitability of a particular resource for a particular task given the values of the metrics. LINKS will develop a simulator in which multiple approaches to formalization of the underlying optimization problem will be tested and the most suitable one will be implemented in the final version of the LEXIS Orchestrator.

4.4 FULFILLING PILOT-DRIVEN REQUIREMENTS

Availability of a checkpointing mechanism is one of the identified requirements coming from the WP5 and WP7 pilot use cases. The mechanism has to be implemented by the application itself, which periodically persists its current state – checkpoint, for example in a file. The application then should be able to load the previous stored state and continue with the execution. Limiting factor can be that the application is able to perform the checkpointing in long periods, therefore the shorter the checkpointing interval is, the less resources are wasted in case of the execution failure.

To support this feature on an HPC cluster, the orchestrator typically has to provide the application a path to a storage in which the application stores its checkpoints. The LEXIS Platform can support this by providing a DDI API used by an application to store a checkpoint which can be later recovered for example in different HPC centre with the access to the LEXIS DDI.

In a Cloud environment, common mechanism for the implementation of the checkpointing can be volume snapshots. The Cloud service such as OpenStack has an API which can be called and snapshot of a particular volume mounted to a VM is created. The LEXIS Orchestrator supports OpenStack API, therefore it satisfies this requirement also in the Cloud.

Other requirements are related mainly to the capability of the orchestrator to manage urgent computing workflows. Their typical characteristics are even-triggered execution (earthquake, flood) and hard requirement for high-availability of the results. The second one can be further divided in a requirement to finish the execution in a certain deadline and to provide resiliency for infrastructure failures.

The event-based execution is supported easily, by introducing a concept of iteration in the workflow, where advance of the iteration is driven by a task which listens to an external data-source for the trigger event. Requirement for the execution deadlines is satisfied by introducing the *dynamic* placement policy, which can be used for the execution for shortest possible run-time given the current load of the resources, availability of reserved queues and so on. High-availability (HA) requirement is satisfied in a similar way, where the same workflow can be executed simultaneously on multiple centres to ensure its completion as well as resilient deployment of the LEXIS Platform itself as described in the following section. The placement policy implemented by the orchestrator can be modified to support such case.

4.5 SERVICE AVAILABILITY LEVEL AND RECOVERY OF WHOLE PLATFORM

The High Availability of LEXIS orchestration system is addressed by having HA deployments for both Yorc (orchestration back-end) and Alien4Cloud (orchestration front-end).

Yorc can have as many instances as needed to scale horizontally. Each instance of Yorc provides a set of workers responsible for executing workflow steps.

Each Yorc instance is stateless:

- Static data like workflow description, associated Ansible playbooks, shell scripts, are stored in a POSIX distributed file system (NFS for example) accessible by all instances.
- Runtime data are stored in a distributed key-value store, HashiCorp Consul [34].

Yorc High Availability is implemented relying on Consul features — service registry, health check and DNS forwarding. Each Yorc instance registers itself as a service within Consul, providing a TCP health check endpoint. Whenever a Yorc instance becomes unavailable, Consul will detect this through its registered Yorc service health check and will stop to resolve the DNS requests to this Yorc instance, allowing seamless fail-over (see Figure 2).

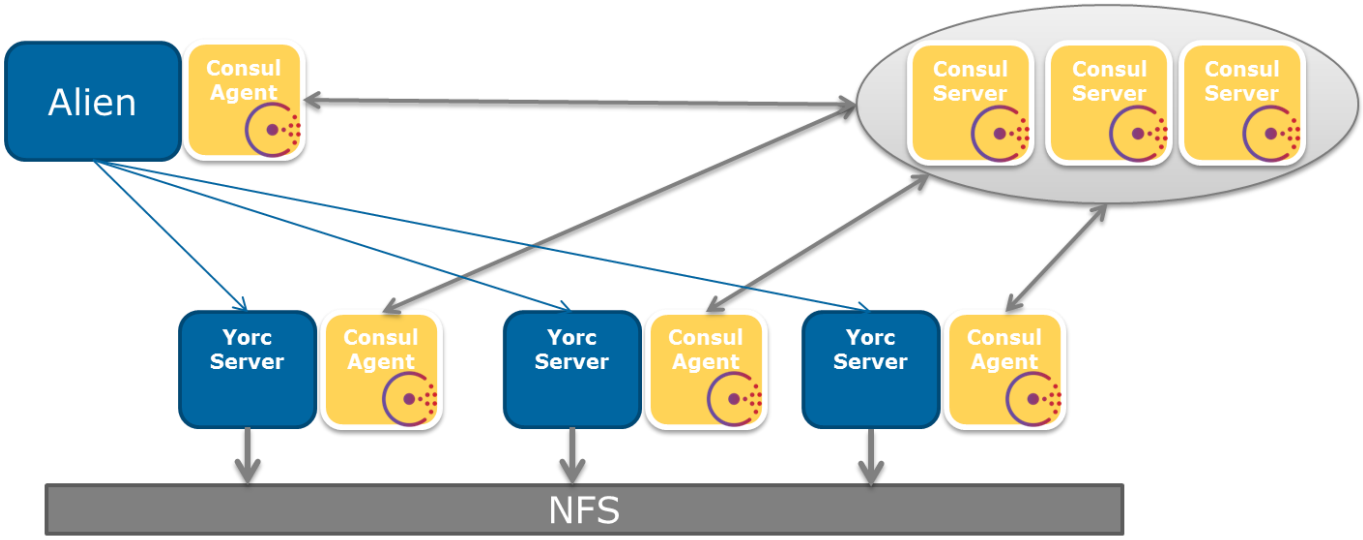

Figure 2 Yorc high availability deployment

The front-end Alien4Cloud is using a DNS domain name to connect to Yorc, this domain name is resolved by Consul to available Yorc instances, as seen above.

An Alien4Cloud HA deployment is based on a primary-backup mechanism (see Figure 3).

Static data like archive of application templates, plugins, are stored in a POSIX distributed file system. Alien4Cloud runtime data are stored in an Elasticsearch cluster.

High availability is implemented using Consul features: health check and leader election. Whenever the primary Alien4Cloud instance health check fails, the backup instance will be elected as the leader and will become active.

A reverse proxy like Nginx can then be used to have a single-entry point for Alien4Cloud instances.

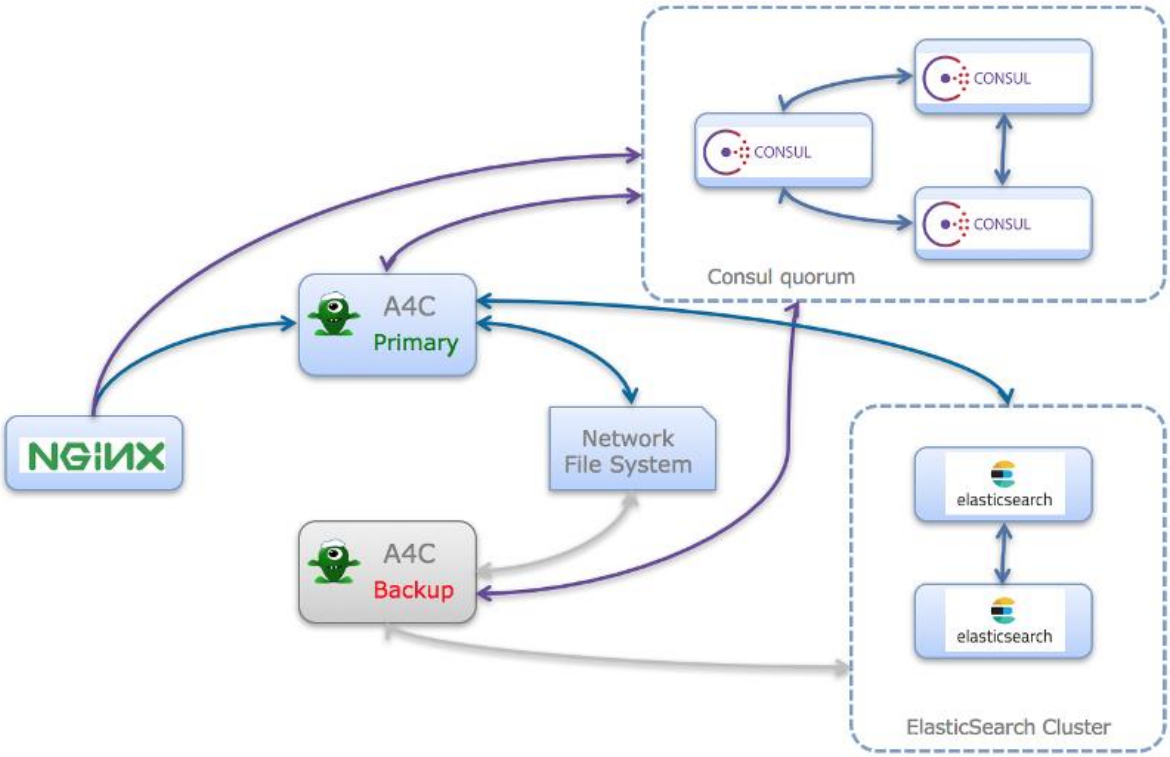

Figure 3 Alien4Cloud high availability deployment

See additional details on High Availability deployments in Yorc documentation [35] and Alien4Cloud documentation [36].

4.6 ORCHESTRATOR SECURITY

As described in Deliverable D4.5 [37], the orchestrator front-end Alien4Cloud requires user authentication. This authentication is delegated to LEXIS AAI that is based on the open source Keycloak solution.

Alien4Cloud communicates with its back-end orchestrator Yorc using TLS (Transport Layer Security) with mutual authentication through certificates.

Yorc itself relies on HEAppE for the authentication required to access HPC and Cloud infrastructures.

5 ORCHESTRATOR GLOSSARY AND TERMINOLOGY

The execution of workflows involves multiple levels of the LEXIS architecture, where executed/managed entities have specific meaning at the different levels. To make the reader aligned with the terminology used all over the LEXIS project and in the documentation, in the following, a set of terms (forming the LEXIS vocabulary) is presented along with their definitions.
Since, the LEXIS architecture is aimed at providing execution resources to the end users, who can be not aware of the underlying technological aspects involved in the LEXIS architecture, the terms are grouped in two sets corresponding to a user-view and to the actual execution environment (*i.e.*, YSTIA-orchestrator view). Figure 4 provides the user-view perspective of the LEXIS platform and how this is involved in the execution of workflows.

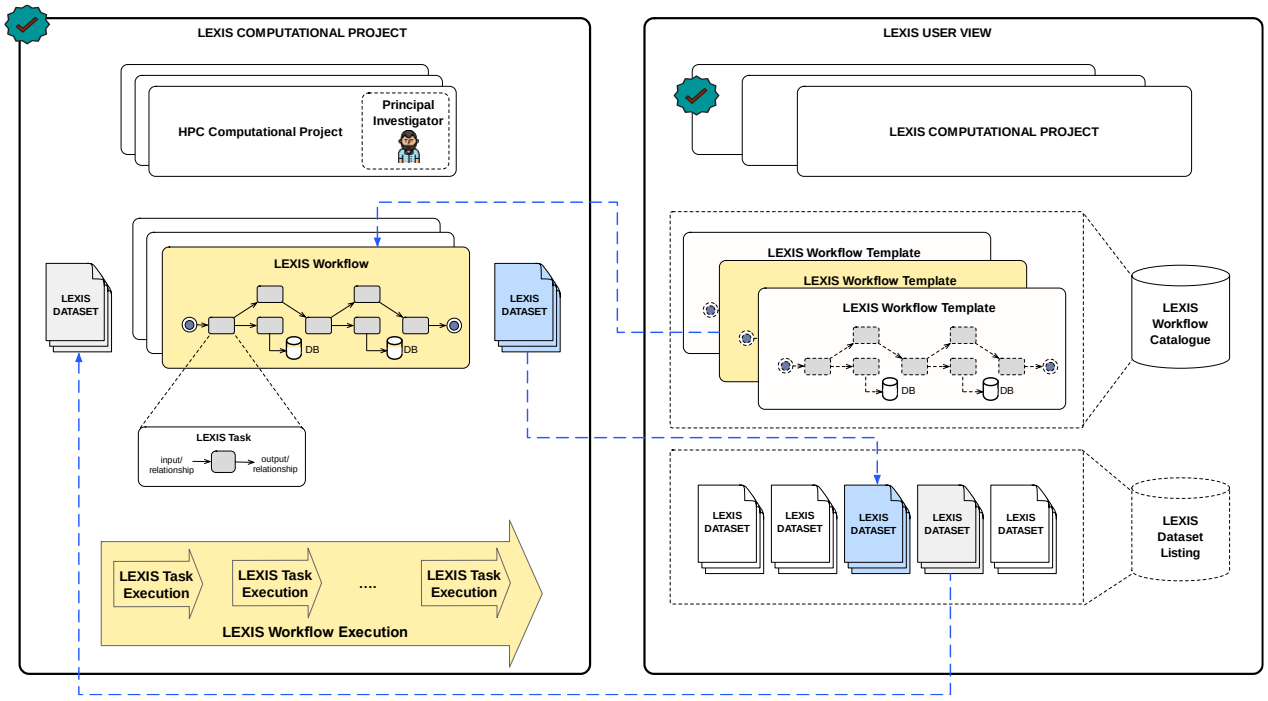

Figure 4 User-view of the LEXIS platform entities and their relationships

5.1 LEXIS ENTITIES TERMINOLOGY

5.1.1 LEXIS Computational Project

A LEXIS Computational Project is an abstraction of the computing and storage resources allocated for running LEXIS workflows. As such, each LEXIS Computational Project may comprise of multiple associated HPC Computational Projects which may be provided by different HPC centres following some dedicated approval process.

5.1.2 HPC Computational Project

An HPC Computational Project is an abstraction of the computing resources allocated to the HPC users by the HPC centres for running their applications. HPC centres usually allocate a given amount of resources represented by core hours (or different unit, depending on the centre). HPC Computational Projects have the following characteristics:

- Requests are submitted by users that want to utilize the resources. HPC centres then grant the resources to the selected requests. Part of the request for the resources is also a selection of a computing resource of the centre (GPU nodes, Cloud, etc.).

- After the resources are granted, the HPC Computational Project is created. The project itself normally has an ID which is unique within a given centre and has a given timeframe in which the resources have to be consumed.
- Such a project typically has one Principal Investigator (see below) and possibly further project members with administrator-like roles (privileged users), depending on the HPC centre and the project. The privileged and normal users within an HPC Computational Project have respective accounts in the HPC centres' AAI systems.

5.1.3 Principal Investigator

Each HPC Computational Project must have a Principal Investigator (PI) who is responsible for ensuring appropriate use of the Project resources. The PI can perform standard administrative operations on the project including adding, removing users, view usage, requesting more resources etc. Note that LEXIS can support HPC Computational Projects in which the PI is not involved in any LEXIS Project (*i.e.*, he/she is not a LEXIS user).

5.1.4 LEXIS Task

A LEXIS Task represents an element of the LEXIS Workflow, and it defines a set of operations to be performed, along with attributes such as the required input data, output data, monitoring information etc. The lifecycle of a LEXIS Task depends on the specific operations that it has to perform; however, it is (in any case) restricted to the lifetime of the LEXIS Workflow.

5.1.5 LEXIS Workflow Template

The LEXIS Workflow Template is a description of the connections of LEXIS Tasks that must be performed to complete the processing of the input LEXIS Datasets and producing output LEXIS Datasets. As such, LEXIS Tasks inside a LEXIS Workflow Template can be organized to form a DAGs (Direct Acyclic Graph), with possibility to access to databases, etc. A LEXIS Workflow Template provides a description of its LEXIS Tasks and the correct sequence of their execution.

5.1.6 LEXIS Workflow

Starting from a LEXIS workflow template, the corresponding LEXIS Workflow associates the inputs to the LEXIS Tasks and the application of configurations (*i.e.*, setting up of specific input parameters) to each LEXIS Task.

5.1.7 LEXIS Workflow Execution

A LEXIS Workflow Execution represents the actual deployment of the LEXIS Workflow on the resources selected by the orchestrator. As such, each LEXIS Task is deployed on the set of computational resources dynamically selected by the orchestrator (according to monitoring data, internal placement policies, users' specified criteria), along with their required inputs (*e.g.*, LEXIS User Dataset). The operations defining the LEXIS Tasks are thus executed. As such, the execution of LEXIS Tasks generates outputs (*e.g.*, output LEXIS User Dataset).

5.1.8 LEXIS Task Execution

A LEXIS Task Execution represents the actual execution of a task within a LEXIS Workflow Execution. As such, the tasks are deployed on the resources dynamically selected by the orchestrator.

5.1.9 LEXIS Workflow Catalogue

The LEXIS Workflow Catalogue provides the repository of LEXIS Workflow Templates managed by the LEXIS platform. It contains the set of of workflow templates (catalogue) that users can instantiate and execute on the

associated compute resources in particular centres (HPC systems, Cloud, etc.) Note that access restrictions apply to the Workflow Catalogue: users can only create LEXIS Workflows based on LEXIS Workflow Templates available to them.

5.1.10 LEXIS User Dataset

A LEXIS Dataset represents the input or output set of data that a LEXIS Workflow respectively consumes or produces during its execution. The datasets can be uploaded/downloaded by the LEXIS Portal user interface.

5.1.11 LEXIS Dataset Listing

The LEXIS Dataset Listing produces a list of the datasets available via the LEXIS Distributed Data Infrastructure (DDI) and via the WCDA (Weather and Climate Data API). In general, this comprises of public datasets, project specific data sets and datasets owned/managed by the user, all hosted on the DDI.

5.2 ORCHESTRATION SERVICE TERMINOLOGY

In the following, a set of definitions specifically related to the entities directly managed by the LEXIS Orchestration Service is provided. These entities are closer to their implementation in the YSTIA-based orchestration service. Where appropriate, definitions are taken from the technologies in which they were developed.

5.2.1 YSTIA (A4C) Catalogue

YSTIA(A4C) Catalogue is an internal YSTIA orchestration system repository for storing YSTIA Application Templates, as well as the TOSCA components that can be used to build the YSTIA Application Templates. Each TOSCA component describes an entity that has to be deployed by the orchestrator, thus it can refer to both a software-framework (*e.g.*, the Docker engine), an application (*e.g.*, a specific Docker container) and a computing resource (*e.g.*, a VM where to install Docker engine). The YSTIA Catalogue is also used to locally store YSTIA Application Templates, either already available on the public YSTIA repository (YSTIA Forge) or defined for the purpose of representing LEXIS workflows. The Catalogue can be populated by importing existing components and Application Templates available on the public YSTIA repository, and by new ones defined by YSTIA architects and application managers (see deliverable D4.2 [9]).

5.2.2 YSTIA (A4C) Application Template

An YSTIA Application Template provides a description of the computational components (*i.e.*, jobs, virtual computing instances, containers, *etc*.) and their relationships, that are intended to be executed on the allocated computational and storage resources. An YSTIA Application Template is stored in the YSTIA catalogue. An YSTIA (A4C) Application Template allows to define input/output parameters, that may be used to specify input/output datasets that are, respectively, consumed and generated once instantiated and executed. The access to the dataset is mediated through the DDI API.

5.2.3 YSTIA (A4C) Application

It is an instantiation of an YSTIA Application Template on the resources selected by the orchestration engine.

5.2.4 YSTIA (Yorc) Workflow

It defines the set of operations that must be performed to execute an Ystia Application. The operations can be executed both sequentially and in parallel, as generally DAGs can be supported. Since YSTIA components are

associated to an interface exposing a set of actions (*i.e.*, start, stop, deploy, run, etc.) that are performed by the orchestrator engine, more than one workflow can be defined and associated to an YSTIA Application Template.

5.2.5 YSTIA (Yorc) Workflow Run

The YSTIA(Yorc) Workflow Run represents the workflow that is executed by the orchestration engine in accordance to run an action. It comprises all the operations required to run jobs on the HPC resources and on the Cloud virtual instances.

5.2.6 LEXIS Workflow Extraction from AC4 Application Template

An YSTIA Application Template provides a detailed description of the components and their relationships, which in turn describe the entire execution process for a given application. The operations required to execute the corresponding YSTIA Application are contained in the YSTIA workflows. The A4C Application Template also serves as a concrete description of a LEXIS Workflow. As such, the LEXIS Workflow Extraction consists in the process of abstracting as much as possible the actions involved in the YSTIA workflow(s), exposing A4C Application Template components which contain tags describing their functionality.

6 SUMMARY

This document provides an initial description of the LEXIS orchestrator allocation policies. First sections summarise requirements and metrics provided by the pilot use cases (WP5-7) and infrastructure consisting of IT4I and LRZ Cloud and HPC systems. Basic algorithmic framework is proposed based on the provided requirements and metrics and possible allocation policies are discussed. Common terminology glossary is also provided to establish a common ground for orchestrator related communication within the project consortium. The discussed policies will be further formalized and verified by simulator developed by LINKS and furthermore by LEXIS Pilot use cases. The improvements will also gain from the requirements and overall progress of the LEXIS Open Call applicants (see Deliverable D9.4 [38]). Formalized description of the policies will be presented in the Deliverable 4.6 [1].

REFERENCES

[1] LEXIS Deliverable, *D4.6 Design and Implementation of the HPC-Federated Orchestration System – Final.*

[2] LEXIS Deliverable, *D5.1 Turbomachinery Use Case: Analysis of Results Run on State-of-Art HPC System.*

[3] LEXIS Deliverable, *D5.2 Rotating Parts Use Case: Analysis of Results Run on State-of-Art HPC System.*

[4] LEXIS Deliverable, *D6.1 Baseline scenarios and requirements.*

[5] LEXIS Deliverable, *D6.2 Pilots Improvements: Solutions Adopted.*

[6] LEXIS Deliverable, *D7.1 Design for Interchange of Weather & Climate Model Output between HPC and Cloud Environments.*

[7] "Intel OPAE," [Online]. Available: https://opae.github.io/latest/index.html. [Accessed May 2020].

[8] A. Rajasekar, R. Moore and et al., "iRODS primer: integrated rule-oriented data system," *Synthesis Lectures on Information Concepts, Retrieval, and Services,* vol. 2, no. 1, pp. 1-143, 2010.

[9] LEXIS Deliverable, *D4.2 Design and Implementation of the HPC-Federated Orchestration System - Intermediate.*

[10] M. Somnath and M. Pranzo, "Power efficient server consolidation for cloud data center," *Future Generation Computer Systems,* vol. 70, pp. 4-16, 2017.

[11] M. C. Cohen, P. W. Keller, M. Vahab and M. Zadimoghaddam, "Overcommitment in Cloud Services: Bin Packing with Chance Constraints," *Management Science,* p. 1–17, 2019.

[12] S. H. Madni, M. S. A. Latiff, S. M. Abdulhamid and J. Ali, "Hybrid gradient descent cuckoo search (HGDCS) algorithm for resource scheduling in IaaS cloud computing environment." Cluster Computing 22.1," *Cluster Computing,* vol. 22, no. 1, pp. 301-334, 2019.

[13] M. Somnath, A. Scionti and A. S. Kumar, "Adaptive resource allocation for load balancing in cloud," in *Cloud Computing*, Springer, Cham, 2017, pp. 301-327.

[14] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," *Journal of grid computing,* vol. 14, no. 2, pp. 217-264, 2016.

[15] R. Prodan, M. Wieczorek and H. M. Fard, "Double auction-based scheduling of scientific applications in distributed grid and cloud environments," *Journal of Grid Computing,* vol. 9, no. 4, p. 531–548, 2011.

[16] G. Iyer and B. Veeravalli, "On the resource allocation and pricing strategies in Compute Clouds using bargaining approaches," in *17th IEEE International Conference on Networks (ICON)*, 2011.

[17] A. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010.

[18] Z. Liu, S. Wang, Q. Sun, H. Zou and F. Yang, "Cost-Aware Cloud Service Request Scheduling for SaaS Providers," *The Computer Journal,* vol. 57, no. 2, pp. 291-301, 2013.

[19] M. Rahman, R. Hassan, R. Ranjan and R. Buyya, "Adaptive workflow scheduling for dynamic grid and cloud computing environment," *Concurrency and Computation: Practice and Experience,* vol. 25, no. 13, p. 1816–1842, 2013.

[20] G. Xu, Y. Ding, J. Zhao, L. Hu and X. Fu, "A novel artificial bee colony approach of live virtual machine migration policy using Bayes theorem (Article ID 369209)," *Science World Journal,* vol. 13, 2013.

[21] X. Song, L. Gao and J. Wang, "Job scheduling based on ant colony optimization in cloud computing," in *International Conference on Computer Science and Service System (CSSS)*, 2011.

[22] A. Quarati and et al., "Scheduling strategies for enabling meteorological simulation on hybrid clouds," *Journal of Computational and Applied Mathematics,* vol. 273, pp. 438-451, 2015.

[23] S. H. Leong, A. Frank and D. Kranzlmüller, "Leveraging e-infrastructures for urgent computing," *Procedia Computer Science,* vol. 18, pp. 2177-2186, 2013.

[24] E. J. Korpela, "SETI@ home, BOINC, and volunteer distributed computing," *Annual Review of Earth and Planetary Sciences,* vol. 40, pp. 69-87, 2012.

[25] A. Tsaregorodtsev and et al., "DIRAC: a community grid solution," *Journal of Physics: Conference Series,* vol. 119, no. 6, p. 062048, 2008.

[26] V. Garonne, A. Y. Tsaregorodtsev and I. Stokes-Rees, "DIRAC: Workload Management System," *Computing in High Energy Physics and Nuclear Physics,* pp. 1041-1044, 2004.

[27] A. Tsaregorodtsev, "DIRAC distributed computing services," *Journal of Physics: Conference Series,* vol. 513, no. 3, p. 032096, 2014.

[28] LEXIS Deliverable, *D2.2 Key parts LEXIS Technology Deployed on Existing Infrastructure and Key Technologies Specification.*

[29] LEXIS Deliverable, *D2.3 Report of LEXIS Technology Deployment - Intermediate Co-Design.*

[30] "Alien4Cloud REST API," [Online]. Available: https://alien4cloud.github.io/#/documentation/1.3.0/rest/overview.html. [Accessed May 2020].

[31] LEXIS Deliverable, *D8.1 First Release of LEXIS Portal (will include report).*

[32] LEXIS Deliverable, *D4.3 Definition of Data Access Priority, Analytics Policies, and Security Assessment.*

[33] A. Scionti, K. Goga, F. Lubrano and O. Terzo, "Scionti, Alberto, et al. "Towards Energy Efficient Orchestration of Cloud Computing Infrastructure." Conference on Complex, Intelligent, and Software Intensive Systems. Springer, Cham, 2018.," in *12th International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2018*, 2018.

[34] "HashiCorp Consul," [Online]. Available: https://www.consul.io/intro/index.html. [Accessed June 2020].

[35] "Yorc High Availability," [Online]. Available: https://yorc.readthedocs.io/en/latest/ha.html. [Accessed May 2020].

[36] "Alien4Cloud High Availability," [Online]. Available: http://alien4cloud.github.io/#/documentation/2.2.0/admin_guide/ha.html. [Accessed May 2020].

[37] LEXIS Deliverable, *D4.5 Definition of Mechanisms for Securing Federated Infrastructures.*

[38] LEXIS Deliverable, *D9.4 Open Call Framework and Stakeholders Engagement on Targeted Large-Scale Pilots - first report.*