Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements

DICE Optimization Tools — Final Version

Deliverable 3.9

Deliverable:	D3.9
Title:	Optimization Tools — Final Version
Editor(s):	Danilo Ardagna (PMI).
Contributor(s):	Danilo Ardagna (PMI), Eugenio Gianniti (PMI), Jacopo Rigoli (PMI), Safia Kalwar (PMI), Giuliano Casale (IMP).
Reviewers:	José Ignacio Requeno (ZAR), Lulai Zhu (IMP).
Type (R/P/DEC):	R & P
Version:	1.0
Date:	**27-07-2017**
Status:	**Final version**
Dissemination level:	Public
Download page:	`http://www.dice-h2020.eu/deliverables/`
Copyright:	Copyright © 2017, DICE consortium — All rights reserved

The DICE project (February 2015-January 2018) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644869

Executive summary

This document presents the status of the activities related to task T3.4 that is the realization of DICE Optimization Tools (**D-SPACE4Cloud**). Further details on T3.4 can be found in D1.1 - State of the art analysis, and D1.2 - Requirements specifications. Deliverable D3.9 is the second of two deliverables (D3.8 and D3.9) reporting the status of the development activities of **D-SPACE4Cloud**. For the sake of the reader's convenience, the context of the tool and its requirements are summarized and updated but we did our best to minimize the overlap between the two documents. **D-SPACE4Cloud** allows the architect to assess the performance and minimize the deployment cost of data-intensive applications against user-defined properties, in particular meeting a priori Quality of Service (QoS) constraints (e.g., deadlines for Hadoop MapReduce or Spark applications under different levels of cluster congestion, or servers utilization for Storm clusters). More precisely, D-SPACE4Cloud takes: 1. a set of DTSM models (one for each candidate deployment), 2. a partially specified DDSM model, 3. the Service Level Agreements (SLAs) to be fulfilled, and 4. a description of the execution environment (list of providers, list of virtual machine types or a description of the computational power available in house) and uses them to generate (via DICE transformation tools) a suitable performance model that is then evaluated and optimized. The optimization consists in finding, through hill climbing, the less expensive cluster configuration able to guarantee SLAs. The architect can analyze the application behavior under different conditions; she/he can, e.g., study pros and cons of public clouds versus private cloud in terms of execution costs.

D-SPACE4Cloud is a distributed software system able to exploit multi-core architecture to execute the optimization in parallel, which encompasses different modules that communicate by means of RESTful interfaces or SSH following the Service Oriented Architecture (SOA) paradigm. In particular, it features a presentation layer (an Eclipse plug-in), an orchestration service (referred to as frontend) and a horizontally scalable optimization service (referred to as backend), which makes use of third-party services as RDBMS, simulators and mathematical solvers. Frontend and backend have been presented in Deliverable D3.8. This deliverable focuses on the Eclipse plug-in (which has been integrated also within the DICE IDE) and the achievements obtained between M18 and M30, which include the support to private clouds, Spark and Storm technologies. This deliverable introduces also the features of the premium version of the tool, which integrates an ad-hoc fast simulator developed in collaboration with the EUBra-BIGSEA project and optimization techniques based on machine learning, which usually allow to speed up the optimization time by reducing the number of simulations to be run.

The extensive analyses proposed at the end of this document (which include also the cost impact analysis of the implementation of privacy mechanisms within the NETF case study) are the conclusions of the activities performed within T3.4.

Copyright © 2017, DICE consortium — All rights reserved

Glossary

AC	Admission Control
CPN	Colored Petri Net
DAG	Directed acyclic graph
DIA	Data-Intensive Application
DDSM	DICE Deployment Specific Model
DPIM	DICE Platform Independent Model
DTSM	DICE Platform and Technology Specific Model
GSPN	Generalized Stochastic Petri net
IDE	Integrated Development Environment
JPA	Java Persistence API
JSON	JavaScript Object Notation
LS	Local Search
M2M	Model to Model transformation
MINLP	Mixed integer nonlinear programming
PNML	Petri Net Markup Language
QA	Quality Assurance
QN	Queueing network
QoS	Quality of Service
SLA	Service Level Agreement
SVR	Support Vector Regression
SWN	Stochastic Well-formed net
UML	Unified Modelling Language
VM	Virtual Machine

Copyright © 2017, DICE consortium — All rights reserved

Contents

Copyright © 2017, DICE consortium — All rights reserved

Copyright © 2017, DICE consortium — All rights reserved

List of Figures

Copyright © 2017, DICE consortium — All rights reserved

List of Tables

List of Listings

Copyright © 2017, DICE consortium — All rights reserved

1 Introduction

The main goal of the DICE EU project is the definition of a quality-driven environment for developing *Data-Intensive Applications* (DIAs) grounded atop Big Data technologies hosted in private or public clouds. DICE offers a UML profile, a workflow, and a set of tools for QoS-aware DIAs development. The end users of the DICE methodology (see DICE Deliverable D2.5) are called to design an application using the DICE profile in a model-driven fashion, selecting the appropriate technologies. The outcome of this process is a set of models that, eventually, undergo a verification and an optimization phase in order to avoid design and security/privacy flaws, as well as to reduce execution costs for given quality guarantees. The **DICE Optimization Tool** (codename **D-SPACE4Cloud**) is, within the frame of DICE project, the component in charge of the design-time optimization process; in a nutshell the rationale of this tool is to support the application designer in identifying the most cost-effective cluster configuration that fulfills some desired quality requirements (e.g., deadlines for MapReduce or Spark applications under different levels of cluster congestion, or servers utilization for Storm clusters).

This document describes the final version of **D-SPACE4Cloud**, which is developed in the framework of WP3 task T3.4 and is published as an open source tool in the DICE GitHub repository.[1] [2,3]

1.1 Objectives

The *goal of WP3* is to develop a quality analysis tool-chain that will be used to guide the early design stages of DIAs and the evolution of the quality of the application once operational data becomes available. The outcomes of the work-package are a set of tools devoted to:

1. simulation-based reliability and efficiency assessment;
2. formal verification of safety properties related to the sequence of events and states that the application undergoes;
3. analysis techniques capable of reasoning about the resources and architecture that DIAs require to meet given quality levels expressed in terms of service-level agreements (SLA).

WP3 defines model-to-model (M2M) transformations that accept as input DPIM, DTSM or DDSM design models defined in tasks T2.1 (*Data-aware abstractions*) and T2.2 (*Data-aware quality annotation*) and produce as outputs the analysis models used by the simulation, quality verification, and optimization tools.

Task T3.4 focuses on the development of the optimization tool to tackle the *resource provisioning* problem for DIAs in the specific context of private and public clouds (see Section 3 for more details).

In particular, **D-SPACE4Cloud** takes as input: 1. a set of DTSM models (one for each candidate deployment), 2. a partially specified DDSM model, 3. the Service Level Agreements (SLAs) to be fulfilled, and 4. a description of the execution environment (list of providers, list of virtual machine types or a description of the computational power available in house) and uses them to generate (via WP3 DICE transformation tools) a suitable performance model that is then evaluated and optimized. The optimization consists in finding, through hill climbing, the less expensive cluster configuration able to guarantee SLAs. The architect can analyze the application behavior under different conditions; she/he can, e.g., study pros and cons of public clouds versus private cloud in terms of execution costs.

This deliverable reports on the research activities carried out in task T3.4 between M19 and M30 and provides an update of the architectural and behavioral description of the initial version of the optimization tool, along with the required inputs and the expected outputs.

This is the second of two deliverables (D3.8 and D3.9) reporting the status of the development activities **D-SPACE4Cloud**. For the sake of the reader's convenience, the context of the tool and its requirements are summarized and updated but we did our best to minimize the overlap between the two

[1] `https://github.com/dice-project/DICE-Optimisation-Plugin`
[2] `https://github.com/dice-project/DICE-Optimisation-Front-End`
[3] `https://github.com/dice-project/DICE-Optimisation-Back-End`

Copyright © 2017, DICE consortium — All rights reserved

documents. In particular, recent work focused on the implementation of: 1) Spark and Storm technologies; 2) private clouds. The extensive analyses proposed at the end of this document (which include also the cost impact analysis of the implementation of privacy mechanisms within the NETF case study) are the conclusions of the activities performed within T3.4.

1.2 Motivation

Originally, companies and other entities set up and provisioned Big Data clusters in house exploiting internal infrastructures; Hadoop was the principal technology on the market and it was installed chiefly on bare metal. Hadoop allowed the execution of one DIA at a time assigning all the available resources to that application. In these circumstances, the execution time only depended on the particular dataset. Soon the scenario changed, to improve utilization, resources were virtualized and the cluster shared among different heterogeneous DIAs. Further, we witnessed a tendency to outsource the resource management and cluster provisioning to the cloud with the aim at overcoming the limits of local facilities, while avoiding large upfront investments and maintenance costs. In this scenario, the problem of predicting and analyzing the performance of a certain DIA turns out to be a real challenge. Indeed, DIAs performance depends on other application running on the cluster at the same time.

Within the DICE framework, the ARCHITECT is the figure in charge of ensuring the application under development is compliant with some user-defined non-functional requirements. To do that, she/he has to consider many factors: the application topology, cluster composition and size, leasing costs of virtual resources, computational power in house, and expected concurrency levels and contention. Certainly, this is far from being a trivial task and it calls for an enabling tool, better if integrated with the design environment. The rationale is, in effect, to provide a tool that is able to assist the ARCHITECT in the definition of expected costs and execution times in realistic scenarios.

DICE adopts simulation to study DIAs specified by means of DTSM and DDSM diagrams. The user specifies non-functional requirements in terms of *deadlines* for MapReduce or Spark applications or servers *utilization* for Storm clusters along with a set of suitable providers and virtual machine (VM) types for each of them. In case of private cloud a description of the available hardware and VMs must be provided as well.

The optimization process is designed to be carried out in an agile way. More precisely, DICE optimization tool fosters an approach whereby cluster optimization can be performed through the DICE IDE that hides the complexity of the underlying models and engines. The IDE allows the user to easily load the required models, launch the optimization algorithm, and track the experiments (running or completed). This eliminates the need for the user to be an expert of software profiling and performance prediction, and simulation techniques on which **D-SPACE4Cloud** is grounded.

1.3 Structure of the document

The structure of this deliverable is as follows. Section 2 recaps and updates the requirements related to the Optimization Tools. Section 3 introduces **D-SPACE4Cloud** and illustrates the scenarios supported to optimize DIA deployments. Section 4 discusses the implementation of **D-SPACE4Cloud**, with particular focus on the Eclipse plug-in that has been implemented in the reporting period and the updates implemented within the parallel local search optimizer, which allowed to reduce the number of simulations required during the search process and to boost the tool performance. Section 5 reports the experimental analyses performed to validate the tool. Section 6 discusses how the tool can support also container-based deployments. Section 7 discusses some concluding remarks.

Moreover, Appendix A and Appendix B provide the Eclipse plug-in installation and usage manuals. Appendix C illustrates the functionality implemented by the JMT PNML pre-processor, which is required if the JMT simulator is used for DIAs performance assessment. Finally, Appendix D provides an update of the mathematical formulation of the optimization problem (and in particular the new formulation for private cloud deployments) whereas Appendix E illustrates the input files used to specify machine learning models that, usually, allows to speed up significantly the optimization time and can be exploited by the premium version of **D-SPACE4Cloud**.

Copyright © 2017, DICE consortium — All rights reserved

2 Requirements and usage scenarios

DICE Deliverable D1.2 [1, 2] presents the requirements analysis for the DICE project. The outcome of the analysis is a consolidated list of requirements and the list of use cases that define the goals of the project.

This section summarizes, for Task T3.4, these requirements and use case scenarios and explains how they have been fulfilled in the final **D-SPACE4Cloud** implementation.

2.1 Tools and actors

As specified in DICE Deliverable D1.2, the data-aware quality analysis aims at assessing quality requirements for DIAs and at offering an optimized deployment configuration for the application. The quality assessment elaborates DIA UML diagrams, which include the definition of the application functionalities and suitable annotations, including those for performance and reliability analyses and verification, and employs the following tools:

- Transformation Tools
- Simulation Tools
- Verification Tools
- Optimization Tools

In particular, the optimization tool, **D-SPACE4Cloud**, takes as input a set of candidate DTSM models (one for each candidate deployment) produced by the application designers, and determines the minimum cost configuration such that the QoS constraints associated with the DIA are met. Moreover, **D-SPACE4Cloud** takes as input also a partially specified DDSM model (including the application deployment diagram) which is updated by the tool with the information of the minimum cost configuration found. The final DDSM can be then transformed by the DICER tool (see DICE Deliverable D2.4) and can trigger the automatic deployment of the optimal configuration found. Note that, the initial DDSM can be the one of the current deployment and can be updated by **D-SPACE4Cloud** with a new optimized solution fulfilling the requirements of a DevOps approach.

In the rest of this document, we focus on the tools related to Task T3.4, i.e., **D-SPACE4Cloud** including its GUI and supporting services. According to the DICE Deliverable D1.2 the relevant stakeholders are the following:

- **ARCHITECT** — The designer of DIAs uses **D-SPACE4Cloud** through the DICE IDE.
- **Optimization Tool** (**D-SPACE4Cloud**) — The tool loads the high-level description of the DIA as well as a description of the execution environment (e.g., profiled VMs, available hardware, list of cloud providers, etc.). This input is used to generate a mathematical model representing the problem to be solved by means of a suitable mixed integer nonlinear programming (MINLP) solver. The so-obtained solution is further optimized through an iterative local search process which relies on GreatSPN, JMT or dagSim[4] tools for a more accurate analysis of the DIA performance.

2.2 Use cases and requirements

The requirements elicitation of D1.2 considers a single use case that concerns **D-SPACE4Cloud**, namely UC3.2. This use case has been revisited and updated during the project execution. In this section, we present an updated description of UC3.2 and the corresponding requirements highlighting how they are tackled and implemented in the final tool version.

UC3.2 can be summarized as follows:

[4] dagSim is an ad-hoc event based simulation tool `https://github.com/eubr-bigsea/dagSim` developed by PMI within the H2020 EUBra-BIGSEA project, which is available with the premium distribution of **D-SPACE4Cloud**.

Copyright © 2017, DICE consortium — All rights reserved

ID:	UC3.3
Title:	Optimization of the deployment of a DIA application
Priority:	REQUIRED
Actors:	ARCHITECT, OPTIMIZATION_TOOLS, SIMULATION_TOOLS
Pre-conditions:	There exists a set of DTSM models (each including an activity and a deployment diagram), one for each candidate deployment and a partially specified DDSM model (including a deployment diagram). Cloud resource costs are stored in the OPTIMIZATION_TOOLS internal resource DB.
Post-conditions:	The ARCHITECT starts specifying the performance constraints for the DIA (in particular, cluster utilization for Storm and response time for Hadoop MapReduce or Spark) and reasons about the optimal resource allocation considering the trade-off between cost and performance requirements. Multiple technologies are analyzed by providing multiple DTSMs throught what-if scenarios (each DTSM is profiled with the service demands estimated on the candidate resource considered as a target configuration).

The requirements listed in [1], revised for the final release, are the followings:

ID:	R3.8
Title:	Cost/quality balance
Priority of accomplishment:	Must have
Description:	The OPTIMIZATION_TOOLS will minimize deployment costs trying to fulfill performance metrics (e.g., MapReduce or Spark execution deadlines, Storm cluster maximum utilization).

ID:	R3.10
Title:	SLA specification and compliance
Priority of accomplishment:	Must have
Description:	OPTIMIZATION_TOOLS MUST permit users to check their outputs against the specified SLAs. If an SLA is violated the tools will inform the user.

The requirements **R3.8** (Cost/quality balance), **R3.10** (SLA specification and compliance) have been fully achieved with the final version of **D-SPACE4Cloud** (at M18 they were completed by 60% and 80%, respectively). Moreover, **D-SPACE4Cloud** implements completely the requirements **R3.9** (Relaxing constraints) and **R3IDE.6** (Loading a DDSM level model), see DICE Deliverable D1.2. The only missing requirement from D1.2 is **R3.11** (Optimization timeout) which is deprecated, since the time out feature is not always implemented by the underlying simulation frameworks.

Copyright © 2017, DICE consortium — All rights reserved

3 Optimization tool overview

D-SPACE4Cloud (DICE Optimization Tool) is the optimization tool integrated in the DICE IDE. It consists of three main components: an Eclipse Plug-in, a frontend and a backend service. Details on the overall architecture and on the internals of each component can be found in Section 4. Suffice to say that the Eclipse plug-in allows the ARCHITECT to specify the input models and performance constraints and transforms the input UML diagrams into the input performance models for the performance solver (GreatSPN, JMT or dagSim). The frontend exposes a graphical interface designed to facilitate the download of the optimization results (which are computed through batch jobs) while the backend implements a strategy aimed at identifying the minimum cost deployment.

Multiple DTSMs are provided as input, one for each VMs considered as a candidate deployment. VMs can be associated with different cloud providers. **D-SPACE4Cloud** will identify the VM type and the corresponding number which fulfill performance constraints and minimize costs. The tool takes as input also a DDSM model which is updated with the final solution found and can be automaticaly deployed through the DICER tool (see DICE Deliverable D2.4).

Moreover, the tool requires as input a description of the execution environment (list of providers, list of VM types or a description of the computational power available in house) and the performance constrains. Input files and parameters can be specified by the ARCHITECT through a wizard which will be described in Section 4.1 and Appendix B.

D-SPACE4Cloud exploits the model-to-model transformation mechanism implemented within the DICE Simulation tool (DICE Deliverable D3.4) to generate a suitable performance model (stocastic well-formed network (SWN) or queueing network (QN) to be used to predict the expected execution time for Hadoop MapReduce or Spark DIAs or cluster utilization for Storm.

Such models, as said, are used to estimate the execution time of the application at hand in meaningful scenarios. For this reason **D-SPACE4Cloud** requires that suitable solvers are installed and accessible. In particular, the final version of the tool is shipped with connectors for two particular third-party tool, i.e., JMT [3] and GreatSPN [4], which can handle QN and SWN models, respectively. Refer to [5] for a detailed comparison of the alternatives. Moreover, the tool supports also the *dagSIM* simulator which has been developed by PMI within the H2020 EUBra-BIGSEA project (see EUBra-BIGSEA Deliverable D3.4). dagSIM is an ad-hoc fast event driven simulator specifically designed to simulate DIA that can be modelled as Directed Acyclic Graphs. dagSIM is less accurate than JMT and GreatSPN but is 10x faster and is available only in the premium license path of **D-SPACE4Cloud**.

The set of third-party tools needed to execute **D-SPACE4Cloud** includes also AMPL [6] or CMPL [7], tools for modeling and handling optimization problems in form of mathematical models. In a nutshell, AMPL and CMPL provide modeling tools and connectors to the most important mathematical solvers (as Cplex, Gurobi, Knitro, CBC, etc)

A MINLP solver is also required to run **D-SPACE4Cloud**. For the experiments reported in this document we made use of Knitro solver but also the or GLPK [8] or COIN-OR [9] open source solvers can be adopted.

The sequence diagram depicted in Figure 1 describes the interactions occurring among the component of DICE optimization tool ecosystem. Through the **D-SPACE4Cloud** Eclipse plug-in the ARCHITECT uploads the DICE UML models along with the performance constraints. UML models are transformed through the DICE simulation tool M2M transformation library into GreatSPN or JMT model format. The set of files obtained in this way are then sent to the frontend component which manages the runs, carrying out them in parallel when possible. The frontend component, in effect, searches for an available backend service instance and oversees the various phases of the optimization. In particular, the first phase consists in generating an initial solution in a fast and effective way. To do this, the backend generates a MINLP model, based on Support Vector Regression (SVR), see Section D.3.1 for each job in the considered scenario, for each provider, and VM type, solves them by means of an external mathematical solver and selects the cheapest configuration able to satisfy the performance constraints. This phase is designed to spawn swiftly an initial solution that could hopefully set the successive local search (LS) phase in a promising position within the space of possible solution (also referred to as *Solution Space*,

Copyright © 2017, DICE consortium — All rights reserved

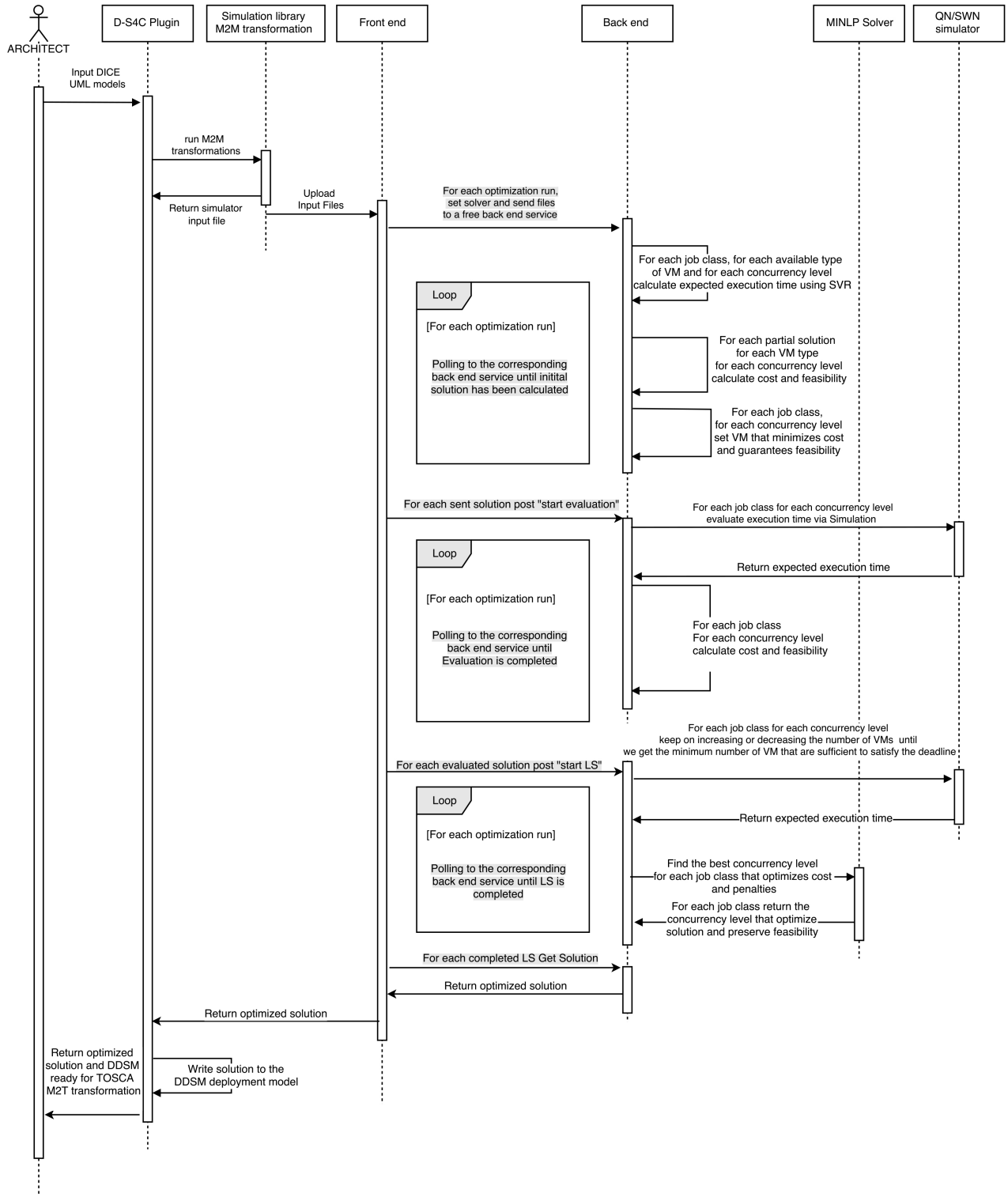


Figure 1: **D-SPACE4Cloud**: Sequence Diagram

Copyright © 2017, DICE consortium — All rights reserved

see Section 4.2). This solution is then evaluated by means of simulation in order to get a more precise (but much slower) estimate of the expected execution times of the jobs considered in the experiment.

Finally, the local search takes the just evaluated solution as a starting point for the optimization process, which essentially consists in increasing/decreasing and changing the VM type for each DIA. This task is executed in parallel where possible, considering SLAs and available resources (in case of private cloud) as constraints. Parallelism is particularly important as each solution spawned during the optimization is evaluated via simulation and this is a time-consuming task. Therefore, in order to make the DICE optimization tool usable we focused on increasing the level of parallelism as much as possible. The final minimum cost configuration found is then returned to the **D-SPACE4Cloud** Eclipse plug-in and is translated into the DDSM which is ready for the TOSCA Model to Text transformation and deployment implemented by the DICER tool.

3.1 Usage scenarios

The aim of this section is to introduce the reader to **D-SPACE4Cloud** usage scenarios. Each scenario corresponds to the analysis of a particular version of the resource allocation problem (formalized in Appendix D) with its own goal, constraints and resolution methods.

Two main analyses can be conducted using **D-SPACE4Cloud**:

- **Public Cloud Analysis** - In this scenario the ARCHITECT wants to analyze the case in which the whole Big Data cluster is provisioned on a public cloud. The first consequence of this choice is that the virtualized resources (i.e., VMs) that can be used to provision the cluster can be considered practically infinite for our purposes. This also means that, under the common assumption that rejecting a job has a much higher cost than the VM leasing costs, it will not apply any job rejection policy in this case. Consequently, the concurrency level for each job, i.e., the number of concurrent user running the job (see Section 4.1) can be set arbitrarily high being always (theoretically) possible to provision a cluster able to handle the load.
  In this scenario, the ARCHITECT may want to know which machine type to select and how many of them in order to execute the application with a certain level of concurrency, meaning considering several similar applications running at the same time in the cluster. She/he might also like to know which cloud provider is cheaper to choose, considering that providers have also different pricing models. For this reason, she/he has to feed the tool with the DTSM diagrams but also with a list of providers, and a list of VM types for each provider. Note that, the information required in the profile of the previous release of **D-SPACE4Cloud** (see DICE Deliverable D3.8), including, e.g., the number of maps, number of reducers, average map time, average reduce time, etc., are now available within the DTSMs (in particular, every DTSM includes such information for each candidate VM type).
- **Private Cloud Analysis** - In this case the cluster is provisioned in house. This choice in some sense changes radically the problem. In fact, the resources usable to constitute a cluster are generally limited by the hardware available. Therefore, the resource provisioning problem has to contemplate the possibility to exhaust the computational capacity (memory and CPUs) before being able to provision a cluster capable of satisfying a certain concurrency level and deadlines. In such a situation the ARCHITECT could consider two sub-scenarios:
  - **Allowing job rejection**, that is consider the possibility to reject a certain number of jobs (lowering consequently the concurrency level), i.e., introducing an Admission Control (AC) mechanism. In this case, since the overall capacity is limited, the system reacts to the excess of load by rejecting jobs; this has an impact on the execution costs as it seems fair to believe that pecuniary penalties can be associate to rejection.
  - **Denying job rejection**, that is imposing that a certain concurrency level must be respected. This translates into a strong constraint for the problem that may not be satisfiable with the resources at hand.

Copyright © 2017, DICE consortium — All rights reserved

Another aspect not to be underestimated is that data-centers are made of physical machines each with a certain amount of memory and cores. **D-SPACE4Cloud** considers the linear relaxation of the problem and estimates the total capacity available as the sum of the memory and number of CPUs (in this way, the underlying optimization problem is a multiple dimensional knapsack). An in dept analysis, discussed in Section 5, has shown that such an approximation is reasonable and it is much faster than a more fine grained modeling (where the underlying optimization problem is a bin packing).
Finally, for what concerns the in house cluster cost function, we consider a cost proportional to the number of physical servers. This simplified cost model is able to capture electricity costs and acquisition costs. If some TCO benchmarks are also considered, the cost model allows to estimate also the data-center operating costs [10].

Copyright © 2017, DICE consortium — All rights reserved

4 DICE Optimization Tool Architecture

The final version of **D-SPACE4Cloud** supports the optimization of Hadoop, Spark, and Storm deployments (an overview of the considered technologies, is provided in the DICE Deliverables D3.4, D3.6, and D3.8). **D-SPACE4Cloud** uses alternatively SWN and QN models to estimate average MapReduce and Spark job completion times and Storm applications throughput and cluster utilization, assuming that the Yet Another Resource Negotiator (YARN) Capacity Scheduler is used.

The optimization model can be applied to both statically partitioned and work conserving mode clusters, but care should be taken in the interpretation of results. In the former case, the performance model provides the mean completion time/throughput for every job class. On the other hand, if the scheduler is configured in work conserving, then the performance metrics we obtain are an approximation due to possible performance gains when resources are exploited by other classes instead of lying idle. SWNs and QN models are described in DICE Deliverables D3.4 and D3.8.

D-SPACE4Cloud follows the Service Oriented Architecture (SOA) pattern. Its architecture (see Figure 2), in fact, encompasses a set of services that can be roughly aggregated in three tiers. The first tier implements the frontend of the optimization tool in form of a standalone Java web service exposing. The frontend is in charge of managing several concurrent optimization runs keeping track of the launched experiments.

The frontend interacts with one or more **D-SPACE4Cloud** backend instances; each instance is a RESTful Java web service in charge of solving the resource provisioning problem introduced in the previous section and formulated in Appendix D. Since the optimization process is a time-demanding operation, the backend has been designed in order to scale horizontally whereas the frontend service is able to balance the load between the backend services.

Finally, the third tier encompasses a set of third-party utilities providing different services (see DICE Deliverable D3.8). In particular, the backend makes use of a relational database (through JPA) to store and retrieve information (e.g., name, memory, number of cores, speed of VM publicly offered by the considered cloud providers). This database is an extension of the resources database developed within the MODAClouds project [11]. Other services in this layer are a mathematical non linear solver, a SWN or QN simulator managed via SSH connection.

In the final **D-SPACE4Cloud** version the frontend is accessed through an Eclipse plug-in which will be extensively described in the next section. More details about the frontend and backend APIs are provided within the DICE Deliverable D3.8, since they have not been changed from the initial release.

The next section overviews the main functionalities of the Eclipse plug-in. Section 4.2 provides an update of the parallel local search optimizer developed within the backend.

4.1 D-SPACE4Cloud Eclipse plug-in

The fist task carried out by the Eclipse plug-in is to provide the user with a simple and intuitive graphical interface to interact with the optimization engine. In particular, it allows to: i) upload the input files defining the problem instance to optimize, ii) launch, stop and cancel the optimization process, iii) download the results of the optimization process.

The main screenshots of **D-SPACE4Cloud** Eclipse plug-in are proposed in Figure 3–Figure 12. The plug-in implements a five step wizard whose windows change according to the selected technology and target deployment. In this section we will provide an overview of the plug-in usage when the Spark technology and public cloud deployment are selected. The complete description is reported in Appendix B.

In particular, Figure 3 shows the initial window of the **D-SPACE4Cloud**. The project window depicts, as an example, a Spark application DTSM model. The optimization tool starts pressing button 1.

In the first step of the wizard (see Figure 4) the user has to specify the number of classes,[5] the DIA target technology and the deployment (public or private). If the public cloud deployment is selected (see

[5] An application class is defined as a set of users runnig the same DIA and characterized by the same SLA constraints, e.g. application execution deadline.

Copyright © 2017, DICE consortium — All rights reserved

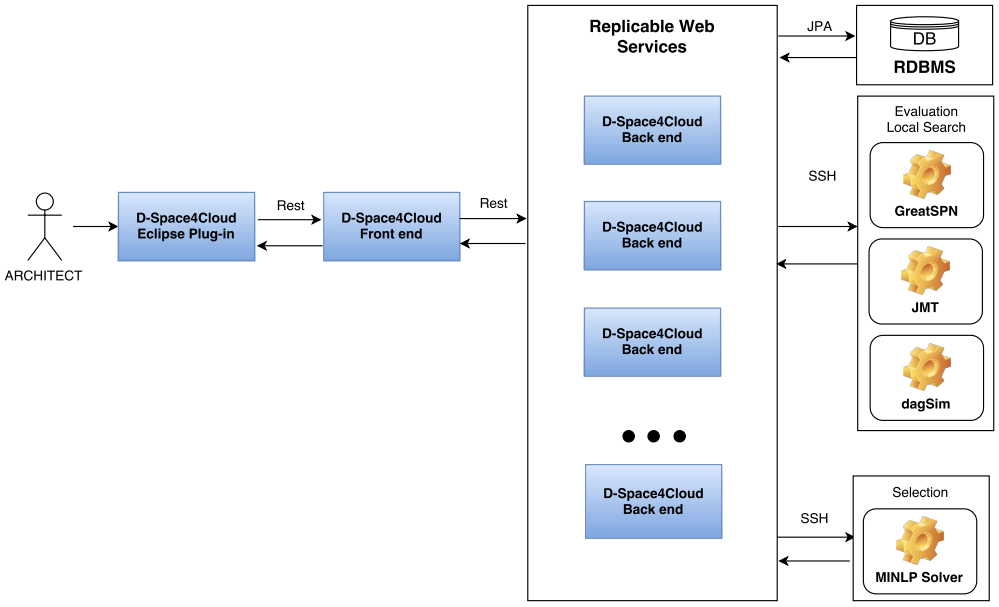


Figure 2: **D-SPACE4Cloud** three-tier architecture

Figure 5) the user has to specify also if there is an existing long term contract (e.g., Amazon reserved instances) or not and, in the former case, specify the number of reserved instances. The premium release of **D-SPACE4Cloud** supports also the use of spot instances (and in this case the fraction, i.e. a number between 0 and 1, of the cluster capacity to be devoted as spot needs to be specified).[6]

Next (see Figure 6), the user has to specify the optimization alternatives selecting, possibly, multiple VM types at different providers candidate for the final deployment. For each VM type, the user needs to select the corresponding DTSM model (see Figure 7) profiled with the service demands expected when the DIA runs on the candidate VM type. The last input of this step, is the DDSM model (see Figure 8), which includes the deployment model of the DIA which will be updated by **D-SPACE4Cloud** with the optimal solution found. Such model can be processed by the DICER tool (see DICE Deliverable D2.4) to obtain the TOSCA description of the optimal configuration, whose automatic deployment can be obtained through the DICE delivery service (see DICE Deliverable D5.3).

Note that, in the **D-SPACE4Cloud** premium version, a machine learning model, which is able to predict the DIA model performance according to its profile characteristics, can be also provided. This significantly speed-up the optimization process providing a very good initial solution for the local search procedure.

The next wizard window allows to input the optimization constraints and it is technology specific. In particular, for Spark and Hadoop MapReduce (see Figure 9), the user can specify the minimum and maximum number of concurrent users, the DIA end users' think time and the DIA deadline (job penalty cost can be specified only on the private cloud case). Note that, for Spark the minimum and maximum number of users needs to be equal to 1 if JMT and GreatSPN simulation tools are used while dagSim supports any number.

When also the optimization constraints are specified, the end user can press the Finish button. Then the window in Figure 10 is shown and the optimization process starts. When the final solution is obtained, the window in Figure 11 is displayed and the results can be downloaded by pressing the main window buttons 2 or 3 (according to public or private deployments). Note that, the DDSM model selected in

[6]We consider a pricing model similar to Amazon EC2, and we assume that IaaS provider offers: 1) reserved VMs, for which the end user applies for a one-time payment (currently every one or three years) for each instance they want to reserve; 2) on-demand VMs, which allow end users to access computing capacity with no long-term commitments; and 3) spot VMs, for which end users bid and compete for unused IaaS capacity. In particular, in this latter case, the end user specifies the maximum price she/he is willing to pay per instance hour. The on-spot price (which is usually significantly lower than on demand or reserved hourly costs) is fixed by the IaaS provider, which can also decide to do not allocate any spot instance to a user. For this reason, spot VMs are considered less reliable.

Copyright © 2017, DICE consortium — All rights reserved

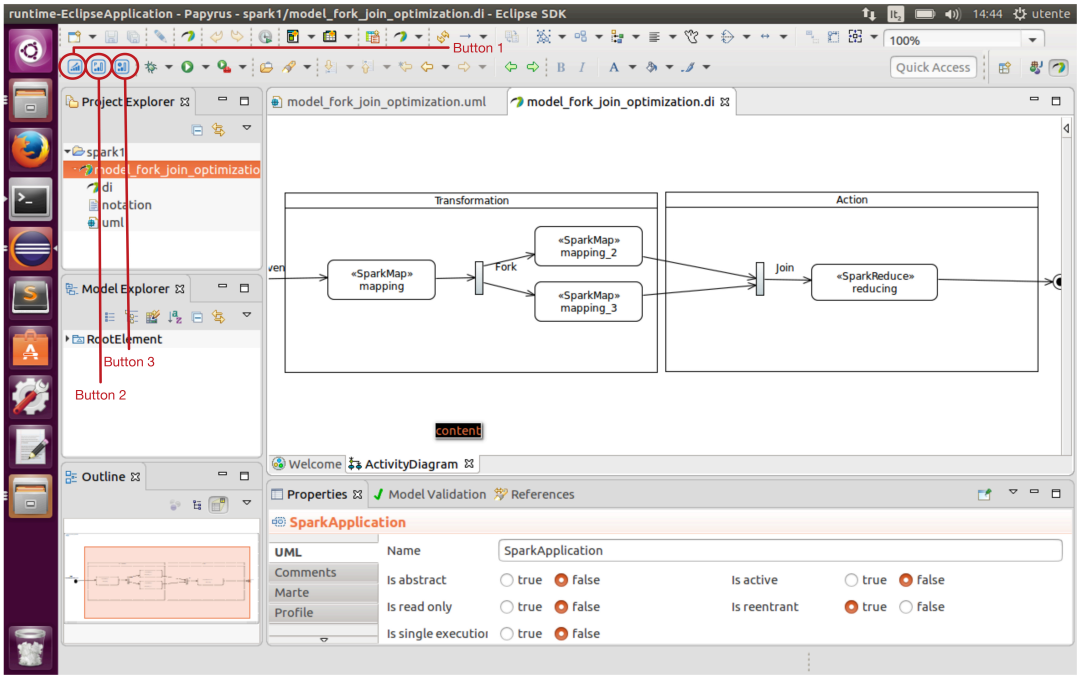


Figure 3: **D-SPACE4Cloud** Main Window

the previous steps will be automatically updated while additional files or information can de obtained through the window in Figure 12 like **D-SPACE4Cloud** start time or the cost of the final solution found. The results window allows also to cancel or restart an optimization process if it failed for any issues and to download the input files and low level output files used by the backend (file format is discussed in DICE Deliverable D3.8).

Copyright © 2017, DICE consortium — All rights reserved

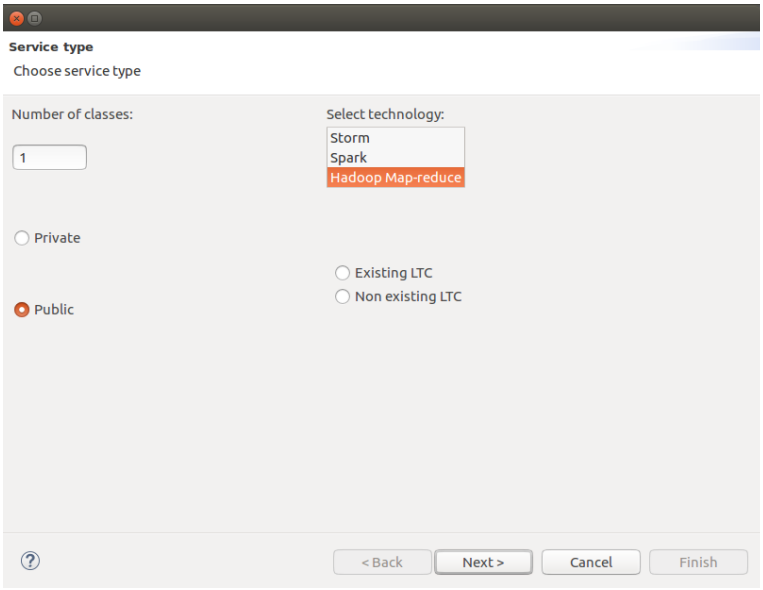


Figure 4: Wizard step 1. Selecting DIA technology and target deployment

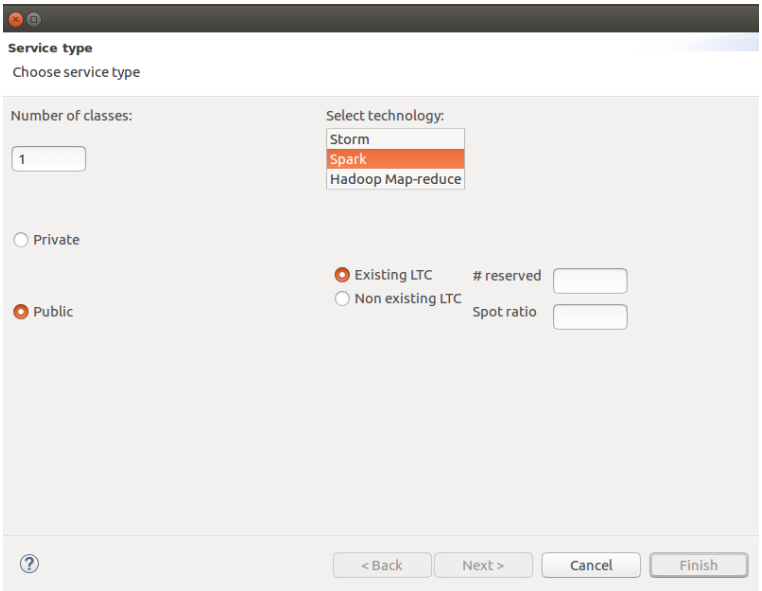


Figure 5: Public cloud deployment with existing long term contract

Copyright © 2017, DICE consortium — All rights reserved

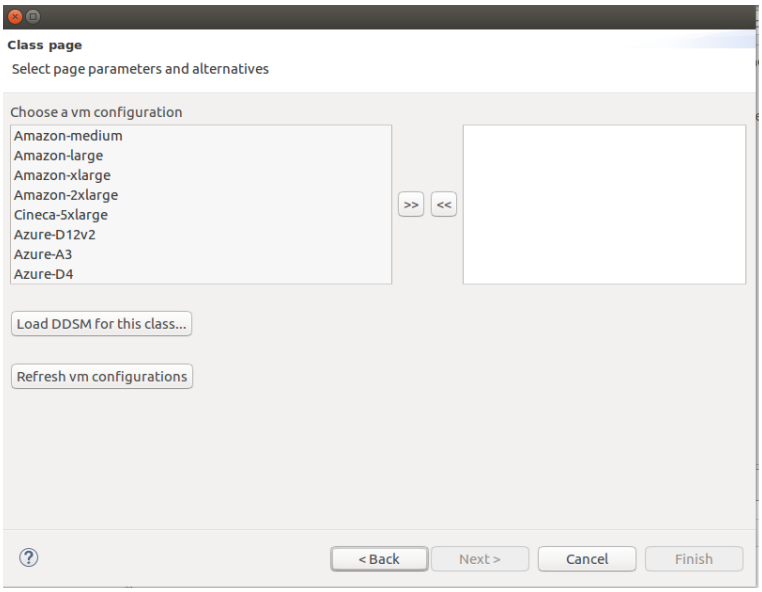


Figure 6: VM type selection and DICE model specification

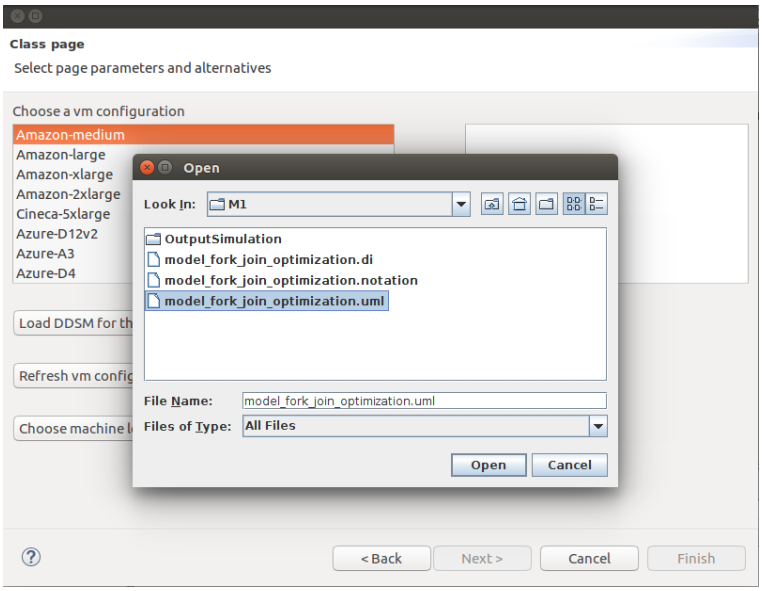


Figure 7: DTSM selection

Copyright © 2017, DICE consortium — All rights reserved

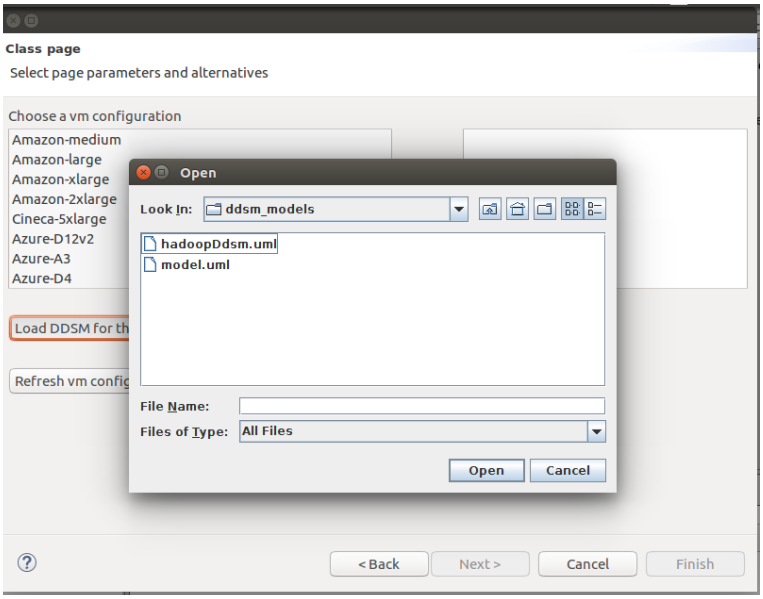


Figure 8: DDSM selection

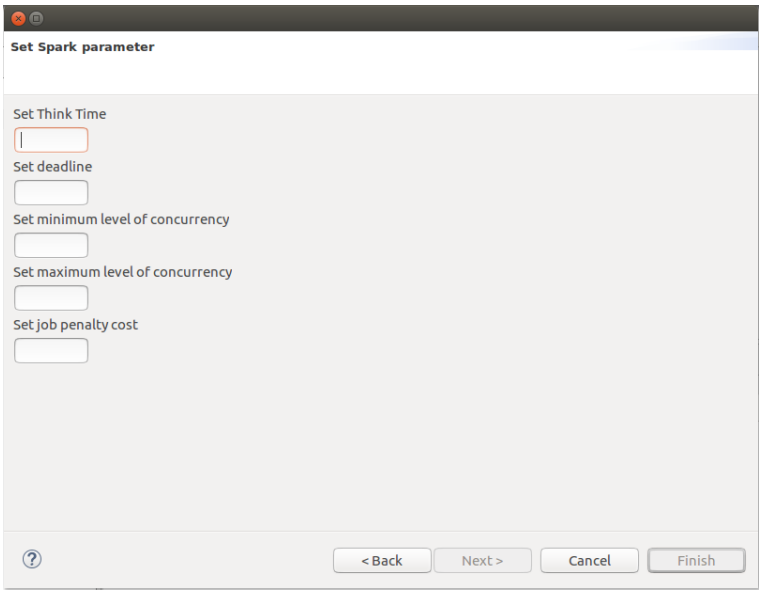


Figure 9: Spark and Hadoop MapReduce optimization constraints

Copyright © 2017, DICE consortium — All rights reserved

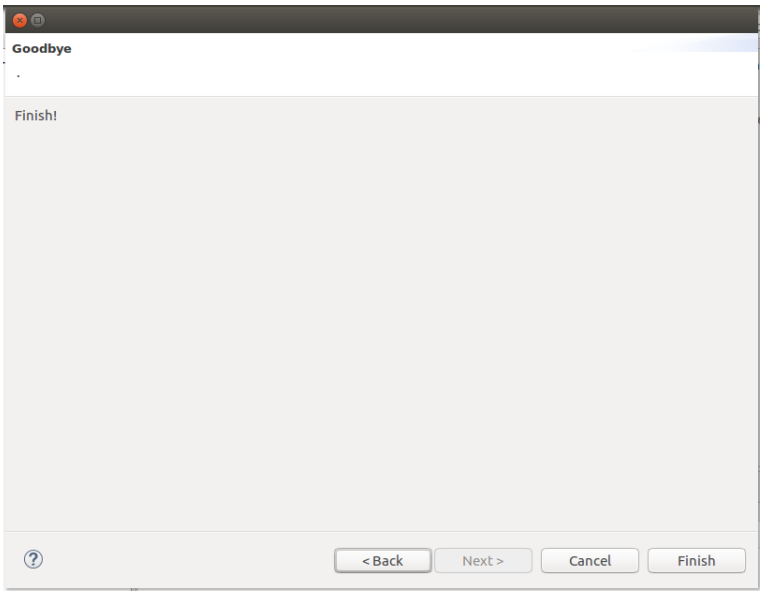


Figure 10: **D-SPACE4Cloud** wizard Finish window

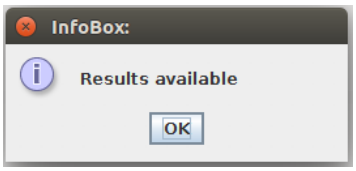


Figure 11: Download window

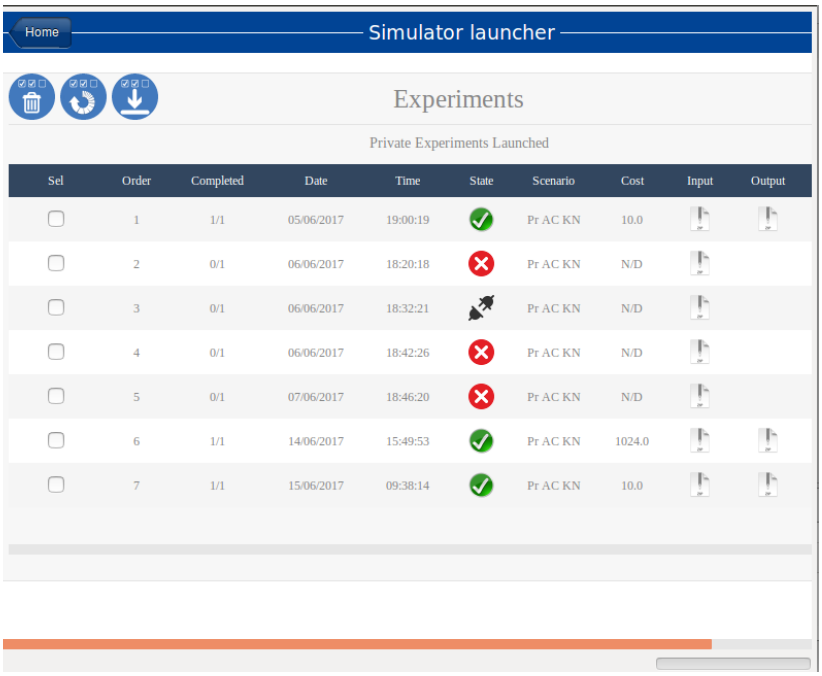


Figure 12: Results window

Copyright © 2017, DICE consortium — All rights reserved

4.2 Parallel Local Search Optimizer

Since the minimum cost deployment optimization problem is NP-hard and the simulation to obtain DIAs performance is a time-demanding operation, the backend has been designed in order to scale horizontally whereas the frontend service is able to balance the load between the backend service instances. A heuristic approach has been adopted and a component called *Parallel Local Search Optimizer* has been devised. Internally, it implements a parallel hill climbing (HC) technique to optimize the number of replicas of the assigned resource for each application; the goal is to find the minimum number of resources to fulfill the Quality of Service (QoS) requirements. This procedure is applied independently, and in parallel, on all application classes and terminates when a further reduction in the number of replicas would lead to an infeasible solution.

As soon as all the classes reach convergence, the DDSM of the final optimal solution found is provided by the Eclipse plug-in, which can then be deployed through the DICER tool.

In particular, HC is a local-search-based procedure that operates on the current solution performing a change (more often referred to as *move*) in the structure of the solution in such a way that the newly generated solution could possibly show an improved objective value. If the move is successful, it is applied again on the new solution and the process is repeated until no further improvement is possible. The HC algorithm stops when a local optimum is found; however, if the objective to minimize is convex, HC is able to find the global optimum. This is the case of the considered cost function, which is linear in the number of VMs in the cluster once the VM type is selected: refer to Appendix D for more details. Hence, every feasible instance of the inner problem can be heuristically solved to optimality via HC.

Algorithm 1 is reported here for clarity purposes. The initial solution S, obtained from the MINLP solution, is evaluated using the QN or the SWN model and each one of its parts is optimized separately and in parallel (line 2). If the partial solution S_i is infeasible the size of its cluster is increased by one unit (line 5) until it reaches feasibility. Otherwise, the procedure attempts to decrease the cost function by reducing the cluster size (line 10). Finally, it is worth pointing out that every time the total number of machines in a cluster is incremented or decremented the best mix of pricing models (i.e., the number of on demand and reserved and spot VMs) is computed so as to minimize the renting out costs of that configuration.

In order to avoid one-VM steps, which might lead to a very long running time for our optimization procedure, particularly when dealing with large clusters, the optimization heuristic exploits the fact that the execution time of Hadoop MapReduce and Spark applications is inversely proportional to the allocated resources [12, 13]. Hence, at every iteration the application execution time is estimated as:

$$r = \frac{a}{\nu} + b, \tag{1}$$

where r is the execution time and ν the number of VMs, whilst a and b are obtained by fitting the hyperbola to previous steps results. Hence, from the second search step on, we can compute a and b using the predicted response times returned by the performance simulators and the associated resource allocations. In this way, at every iteration k it is possible to have a hint on the number of VMs required to meet the deadline D, as depicted in Figure 13:

$$\nu^{k+1} = \frac{a^{k,k-1}}{D - b^{k,k-1}}. \tag{2}$$

Equation (2) is exploited by the procedures ***IncrementCluster*** and ***DecrementCluster*** at lines 5 and 10. Now, the configuration obtained from (2) must be verified via the performance model. Furthermore, we are only using the two most recent simulation results to fit a and b, hence the prediction might possibly oscillate, thus hampering the algorithm convergence. In order to avoid this issue, we keep track of the least VMs feasible solution and, symmetrically, the infeasible one deployed on most VMs. If (2) yields a configuration laying out of this range, the procedures ***IncrementCluster*** and ***DecrementCluster*** fall back to an ordinary binary search and explore the mid point. Note that, the final increment at

Copyright © 2017, DICE consortium — All rights reserved

Algorithm 1 Hill climbing algorithm

Require: $S = \{S_i \mid i \in \mathcal{C}\}$

1: ***Evaluate*** (S)
2: **for all** $i \in \mathcal{C}$ **do** ← *Parallel for*
3: **if** S_i is infeasible **then**
4: **while** S_i is infeasible **do**
5: ***IncrementCluster*** (S_i)
6: ***Evaluate*** (S_i)
7: **end while** } *Pursuit of feasibility*
8: **else**
9: **while** S_i is feasible **do**
10: ***DecrementCluster*** (S_i)
11: ***Evaluate*** (S_i)
12: **end while** } *Cost optimization*
13: ***IncrementCluster*** (S_i)
14: **end if**
15: **end for**
16: **return** S

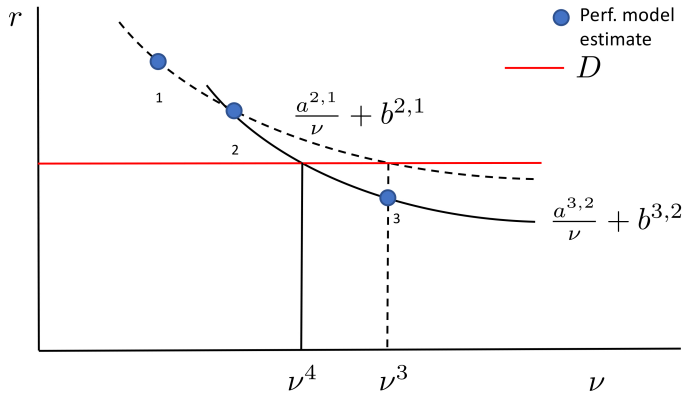


Figure 13: Hyperbolic jump

line 13 is needed, since the while loop at lines 9-12 exits with an unfeasible solution and step 13 allows to roll back to a feasible state.

Thanks to this behavior, the interval keeps shrinking until we find a pair of configurations that differ by only one VM, so that one is feasible and the other is not. Hence the final solution returned at line 16 is provably the minimum size feasible configuration for the submitted application.

The new release of the parallel local search optimizer support also the private cloud case for Hadoop MapReduce and Spark technologies.[7] When an in house cluster is considered, the separation among different pricing models disappears (e.g, reserved or on-demand), but other facility costs should be taken into account. Moreover, in order to contemplate the possibility to exhaust the cluster computational capacity, these costs comprehend also penalties associated to job rejections. This is the Admission Control (AC) case described in Section 3 (the denying job rejection case is trivial and can be reduced to the AC case where the minimum and maximum concurrency levels are equal). Furthermore, AC enlarges the feasible region of the resource provisioning problem by lowering the concurrency level.

Within the DICE project, two different models supporting the private cloud with AC, have been considered. Both models aim to find the optimal concurrency level for each class, respecting the DIA QoS

[7]Note that, for Spark dagSim should be used as the underlying simulator, hence this feature is available only with the **D-SPACE4Cloud** premium version.

Copyright © 2017, DICE consortium — All rights reserved

constraints. Both models minimize the deployment cost, but, the first model (a multiple dimensional knapsack) is simpler and does not consider how to distribute the required VMs among the physical nodes. In contrast, the second model, based on bin packing, takes into account also this fact, generating a more accurate solution. The model formulations are reported in Appendix D. An extensive quantitative analysis is provided in the next section. Since, the solution of the bin packing problems requires significantly more time than the knapsack and the accuracy gain is limited to few percentage figures, the final **D-SPACE4Cloud** release relies only on the multiple knapsack formulation.

Algorithm 2 Private Cloud Algorithm

1: **for all** $i \in \mathcal{A}$ **do**
2: **for all** $l \in \mathcal{H}_i$ **do**
3: concurrency level $\leftarrow l$
4: apply Algorithm 1
5: compute C_{il}
6: **end for**
7: **end for**
8: solve knapsack (or bin packing) problem

The solution algorithm is reported in Algorithm 2. In private clouds if job-rejection is taken into account, the execution cost have to be calculated for all the concurrency levels. Let us denote with i the application class and with $\mathcal{H}_i$ the set of possible concurrency levels (i.e., all the possible numbers of concurrent users which can be obtained with the interval specified in Figure 9). For every pair (i, l), the costs C_{il} can be obtained applying Algorithm 1. Once the C_{il} have been obtained, the results are fed as parameters to the knapsack problem (see Section D.5.1). This is in charge of satisfying the physical cluster capacity constraint and identifying the minimum cost configuration, providing both the optimal type and number of VMs to support the DIA execution and the number of concurrent users.

Copyright © 2017, DICE consortium — All rights reserved

5 Optimization Tool Validation

In this section we show the results of several experiments performed to validate **D-SPACE4Cloud**. All the experiments have been performed on two Ubuntu 14.04 VMs and one Fedora 20 VM hosted on an Intel Xeon E5530 2.40 GHz equipped server. The first VM ran the **D-SPACE4Cloud** web service and KNITRO 10.0 [14], a solver for the mathematical programming language AMPL [6], which was used to solve the optimization problem presented in Appendix D, determining an initial solution to our HC algorithm. This VM hosted also dagSim,[8] an *ad-hoc* simulator for the performance analysis of Big Data jobs which can be represented as directed acyclic graphs (DAGs). The second one, instead, ran JMT 0.9.3 [3], used to simulate QNs while the Fedora VM hosts GreatSPN adopted to simulate Stochastic Well Formed Nets (SWNs).

The results reported in this deliverable refer to Spark public cloud and Hadoop MapReduce and Spark private cloud deployments. An extended validation of Hadoop MapReduce on public clouds is available in DICE Deliverable D3.8. An extended evaluation of Storm applications design is reported in DICE Deliverables D3.3 and D3.4. JMT and GreatSPN accuracy for MapReduce job is also discussed in DICE Deliverables D3.4, D3.8, and [5]. dagSim accuracy has been evaluated within the EUBra-BIGSEA project. The average error for dagSim models settle at 3.56% while the worst case error is 25.16% (see [15] for further details).

The next section presents the experimental settings. Section 5.2 focuses on public cloud scenarios. Private cloud deloyments are analysed in Section 5.3. The input files for most of the analyses reported in this deliverable are available at `https://zenodo.org/record/835569#.WXrf2sZ7G7p`.

5.1 Experimental Setup and Design of Experiments

To profile DIAs and compare **D-SPACE4Cloud** results, we collected real measures by running SQL queries on Apache Hive [16] and Spark [17]. We used the industry standard TPC-DS [18] benchmark dataset generator to create synthetic data at scale factors ranging from 250 GB to 1 TB.

Figure 14 shows the considered queries. R1, R3, and R4 are handcrafted so as to have exactly one map and one reduce stage when run on Hive, thus constituting an example of MapReduce job. On the other hand, Q26 and Q52 are two of the queries that make integral part of the TPC-DS benchmark. These queries have been executed on Spark, yielding complex DAGs of stages, as shown in Figure 15.

Since profiles collect statistical information about jobs, we repeated the profiling runs at least twenty times per query. Properly parsing the logs allows to extract all the parameters composing every query profile, for example average and maximum task execution times, number of tasks, etc. Profiling has been performed on Amazon EC2, by considering m4.xlarge instances, on Microsoft Azure, with A3 or D12v2 VMs, and on PICO,[9] the Big Data cluster offered by CINECA, the Italian supercomputing center. The cluster rented on EC2 was composed of 30 computational nodes, for a total of 120 vCPUs hosting 240 containers, whilst on PICO we used up to 120 cores configured to host one container per core. On the other hand, on Azure we provisioned clusters of variable sizes, reaching up to 26 dual-core nodes. In the EC2 case every container had 2 GB RAM and on Cineca 6 GB, while on Azure containers where 2 GB for A3 machines, 8 GB for D12v2. Along with profiles, we also collected lists of task execution times to feed into the replayer in JMT service centers or dagSim stages. In the end, we recorded the different VM types characteristics.

Private cloud analyses consider the MapReduce R1, R3, R4 queries on the TPC-DS benchmark with 1TB scale factor. Moreover, we introduce query Q1, which is query R1 ran on top of Spark (the corresponding DAG of stages is reported in Figure 16). The deadlines associated to the experiments are extracted from a uniform distribution in the range from 7 to 25 minutes.

Since Big Data are typically in charge of supporting high level decisions, the number of concurrent users that exploit the cluster varies from 5 to 20 as considered in [5]. In order to take into account the rejection of some jobs, each class has also a lower bound on the number of concurrent users fixed to 80%

[8] `https://github.com/eubr-bigsea/dagSim`
[9] `http://www.hpc.cineca.it/hardware/pico`

Copyright © 2017, DICE consortium — All rights reserved

```
select avg(ws_quantity),
       avg(ws_ext_sales_price),
       avg(ws_ext_wholesale_cost),
       sum(ws_ext_wholesale_cost)
from web_sales
where (web_sales.ws_sales_price between 100.00 and 150.00) or (web_sales.ws_net_profit
between 100 and 200)
group by ws_web_page_sk
limit 100;
```

(a) R1 and Q1

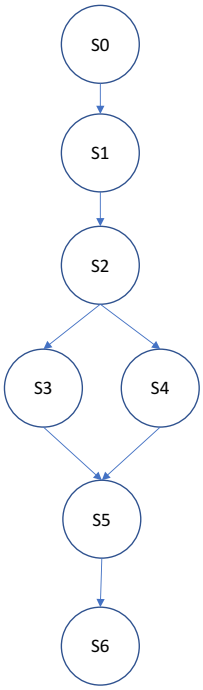
```
select avg(ss_quantity), avg(ss_net_profit)
from store_sales
where ss_quantity > 10 and ss_net_profit > 0
group by ss_store_sk
having avg(ss_quantity) > 20
limit 100;
```

(b) R3

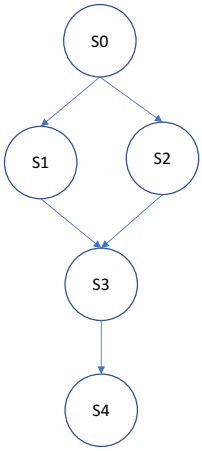
```
select cs_item_sk, avg(cs_quantity) as aq
from catalog_sales
where cs_quantity > 2
group by cs_item_sk;
```

(c) R4

Figure 14: Queries



(a) Q26



(b) Q52

Figure 15: Spark queries DAGs

Copyright © 2017, DICE consortium — All rights reserved

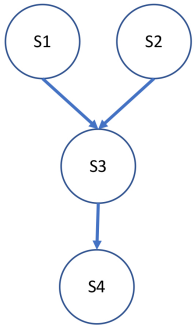


Figure 16: Q1 DAG

of its required maximum concurrency level. Hence, the minimum concurrency level spans from 4 to 16 users. The system reacts to the excess of load by rejecting jobs: this has an impact on the execution costs since pecuniary penalties are associated to rejections. Thus, for each class, its minimum concurrency level is used to determine the maximum number of jobs that can be rejected. When the capacity of the cluster is not large enough to accommodate the minimum required workload, the problem becomes infeasible. The subsequent diagrams highlight solutions that rejected some jobs using red triangles.

The experiments considers two reference types of VMs deployed in the private cluster, named *xlarge* and *5xlarge*, whose parameters are reported in Table 1.

Table 1: Parameters of the considered VMs

Parameter	Value	Units of Measurement
	xlarge	
$core$	5	$[-]$
$memory$	16	$[GB]$
$cost$	0.25	$[e/h]$
	5xlarge	
$core$	20	$[-]$
$memory$	120	$[GB]$
$cost$	1.25	$[e/h]$

Moreover, in order to compare the costs resulting from all the performed analyses, only one type of physical servers has been taken into account. Thus, all the homogeneous servers have the same characteristics, i.e, E = 1.25, M = 120, and V = 20, where E is the cost to run one physical machine, M and V are the servers capacity in RAM and number of CPUs.

Private cloud analyses exploit four different classes whose relevant parameters are summarized in Table 2 (p_i denotes the penalty costs associated with job rejection, H_i indicates the concurrency level while D_i is the deadline associated to class i).

The approach used to estimate the cost of a VM and penalties, which appears in the reported table, is discussed in Section 5.3.1 and Section 5.3.2.

Copyright © 2017, DICE consortium — All rights reserved

Table 2: Parameters of the considered Classes

Query	type of VM	$p_i[e/h]$	H_i (range)	D_i (range)
$R1$	*5xlarge*	6	4 – 20	$7' - 25'$
$R3$	*5xlarge*	14	4 – 20	$7' - 25'$
$R4$	*5xlarge*	15	4 – 20	$7' - 25'$
$Q1$	*xlarge*	3	4 – 20	$7' - 25'$

5.2 Public Cloud Analysis

The goal of this section is to provide the results of some analyses, which show the effectiveness of our solution algorithms for public clouds. Section 5.2.1 reports the results of some relevant scenarios, where the parameters that govern the problem solution are varied one at the time. Section 5.2.2 evaluates the quality of the optimal solution found by **D-SPACE4Cloud** considering a real deployment. Finally, Section 5.2.3 illustrate the preliminary results we achieved for the NETF case study.

5.2.1 Scenario-based Experiments

The optimization approach needs to be validated, ensuring that it is capable of catching realistic behaviors we expect of the system under analysis. We test this property with a set of assessment runs where we fix all the problem parameters but one and verify that the solutions follow an intuitive evolution.

The main axes governing performance in Hadoop or Spark clusters hosted on public clouds are the level of concurrency and the deadlines. In the first case, increasing the number of concurrent users H_i and fixing all the remaining parameters, we expect a need for more VMs to support the rising workload, thus leading to an increase of renting out costs. On the other hand, if at fixed parameters we tighten the deadlines D_i, again we should observe increased costs: the system will require a higher parallelism to shrink response times, hence more computational nodes to support it.

For the sake of clarity, we performed single-class experiments: considering only one class per experiment allows for an easier interpretation of the results.

Figures 17 and 18 show the behavior with single user Spark runs. Q26 exhibits a straightforward behavior, with D12v2 instances always leading to cheaper deployments. In order to quantify monetary savings, we compute the ratio of the difference between costs over the minimum cost alternative. With this metric, D12v2 yields an average percentage saving around 76% for Q26, hence this VM type proves to be the cheapest choice by a considerable margin. On the other hand, Q52 provides a more varied scenario. As shown in Figure 19 for clarity, both VM types cover the role of cheaper alternative when the deadline varies. The configurations where A3 is cheaper, marked with crosses in figure, yield an

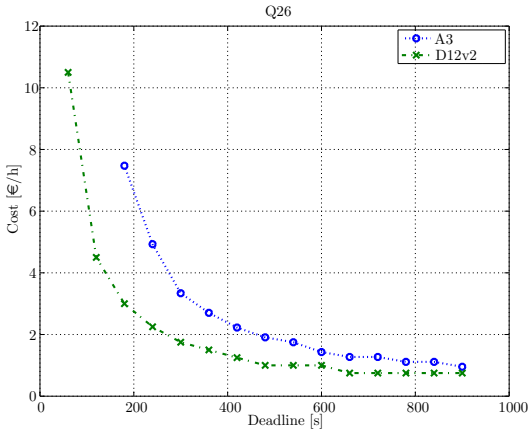


Figure 17: Query Q26, single user

Copyright © 2017, DICE consortium — All rights reserved

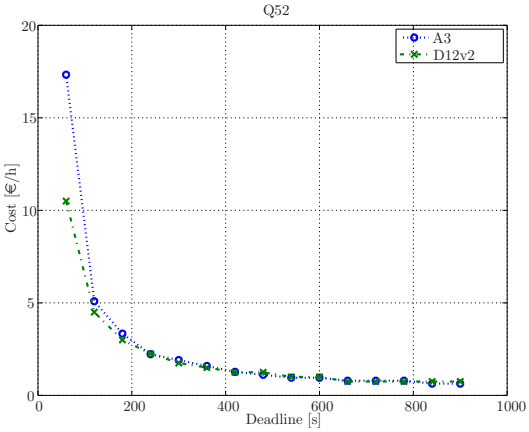


Figure 18: Query Q52, single user

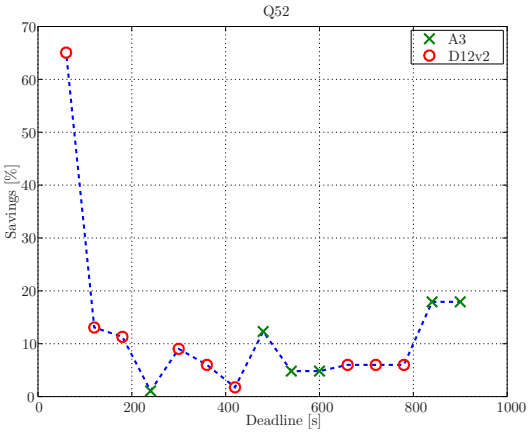


Figure 19: Query Q52, single user, savings

Copyright © 2017, DICE consortium — All rights reserved

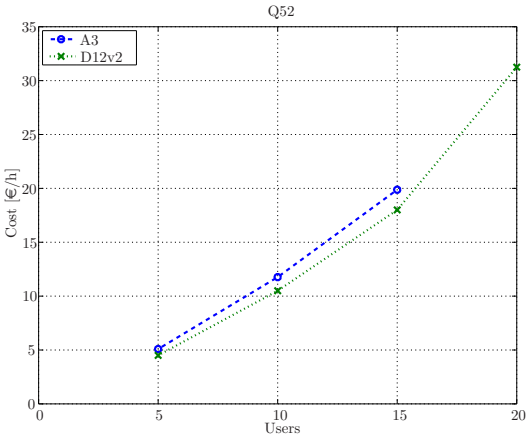


Figure 20: Query Q52, multi-user

average saving of 9.8%, whilst the circles related to D12v2 give a mean saving of 13.8%. The maximum saving is about 65.1%.
Overall, these results provide a strong point in favor of the optimization procedure implemented in our tool, as they prove that making the right choice for deployment can lead to substantial savings throughout the application life cycle.

In terms of execution times, **D-SPACE4Cloud** carried out the whole optimization procedure within minutes. All the running times where in the range $[24, 560]$ s, with an average of 125.98 s.

Figure 20 shows the results of a multi-user experiment. Fixing all the other parameters, in particular query Q52 and a deadline of 10 minutes, we varied the required concurrency level between 5 and 20 users with step 5. Here the D12v2 instances prove in every case the better choice, with an average saving of 11.84%. No feasible solution can be found for 20 users considering A3 instances.

5.2.2 Optimal Solution Validation in a Real Cluster Setting

A further experiment we performed was aimed at assessing the quality of the optimal solution obtained using **D-SPACE4Cloud**. Given a query and a deadline to meet, we focus on the response time measured in a real cluster provisioned according to the number of VMs determined by the optimization procedure, quantifying the relative gap as a metric of the optimizer accuracy. Formally:

$$\varepsilon = \frac{D - R}{D}, \tag{3}$$

where D is the deadline and R the response time, so that possible misses would yield a negative result.

We considered six cases, varying deadline and query, and ran **D-SPACE4Cloud** to determine the optimal resource assignment to meet the QoS constraints on D12v2 instances. Table 3 collects the data that relate to this experiment. Every row shows the relevant query and deadline, the optimal number of VMs, the measured response time, and the percentage gap with respect to D. First of all, none of the considered runs led to a deadline miss. Moreover the relative error is always below 30%, with a worst case result of 26.92% and the average settling at 20.51%. Overall we can conclude that the optimization tool is effective in identifying the minimum cost solution at design time, guaranteeing that deadlines are met as well.

5.2.3 NETF Case Study

To validate **D-SPACE4Cloud** we also considered the NETF case study with the aim to evaluate the cost impact of the implementation of some privacy mechanisms on the taxpayers data. More details are availabel in DICE Deliverable D6.3.

Copyright © 2017, DICE consortium — All rights reserved

Table 3: Optimizer validation

Query	D [ms]	# VMs	R [ms]	ε [%]
Q26	$180,000$	12	$158,287$	12.06
Q52	$180,000$	12	$150,488$	16.40
Q26	$240,000$	9	$186,066$	22.47
Q52	$240,000$	9	$175,394$	26.92
Q26	$360,000$	6	$280,549$	22.07
Q52	$360,000$	6	$276,790$	23.11

```
select  d1.taxpayer,
d1.taxDeclaration,
d1.declarationDate,
d1.income,
d1.otherIncome,
d1.charge,
d1.unemployed,
d1.pension,
d1.taxablePension,
d1.invalidityPension,
d1.alimentaryPension
from Declare d1;
```

(a) Query 5

```
select  dic.id,
                d1.taxDeclaration,
                d1.declarationDate,
                d1.income,
                d1.otherIncome,
                d1.charge,
                d1.unemployed,
                d1.pension,
                d1.taxablePension,
                d1.invalidityPension,
                d1.alimentaryPension
from Declare d1
inner join Dictionary dic on dic.id=d1.taxpayer;
```

(b) Query 6

Figure 21: NETF Queries

Copyright © 2017, DICE consortium — All rights reserved

The queries reported in Figure 21 have been implemented in Spark 2.0 and ran on Microsoft Azure HDInsight on D12v2 VMs with 8 cores and 28GB of memory. The number of executors per node (virtual machine) were either two or four. We set the maximum executor memory 8 GB as for the driver memory while executor were assigned two or four cores. The number of nodes varied between 3 and 12. Overall we ran experiments on up to 48 cores. Runs were performed considering the default HDInsight configuration. In the following, the baseline Query 5 and its anonymised versions will be compared considering the same configuration. Each query for each configuration was run 10 times to evaluate the average execution time and obtain the performance profile.

To achieve anonymization, the first privacy mechanisms applied is masking. For that purpose, a dictionary table has been introduced, which implements a one-to-one mapping between a clear ID and a masked ID. In other words, the tables in the taxpayers database store masked IDs for the taxpayers, while the clear ID is available only in the dictionary table which has a restricted access. Query 6 is derived from Query 5. The idea behind Query 6 is to measure the overhead on the system due to the additional join which is required to read the clear IDs. The second privacy mechanisms considered is encryption with AES at 128 and 256 bit. In this case, the IDs and sensitive data stored in the tables are encrypted and decryption is performed contextually while running Query 5.

Figure 22 compares the different privacy mechanisms for Query 5 on 1.5 millions taxpayers data set. Unfortunately, Query 6 was always failing when more than 6 nodes are used. From Figure 22 , it is evident that masking/join has the maximum effect on the performance than other privacy techniques, while encryption has negligible performance effects. The yellow line (top most line) denotes the execution time for masking, it has around 28% performance degradation while the red and blue line overlaps the plain Query 5 execution time. Encryption has only 3% overhead at maximum.

To evaluate the cost impact of privacy, we considered Query 5 and 6 at 1.5 millions dataset with 85s initial deadline. Then, deadline was iteratively decreased by 20s in order to consider 10 optimization instances. The result are reported in Figure 23 .

From the experimental results one can see that above 45s and for 20s no extra costs are incurred, while for 25 and 15s deadline the cost overhead due to masking technique is in between 50 and 66%. Deadlines lower than 15s resulted to be too strict and D-SPACE4Cloud did not find any feasible solution.

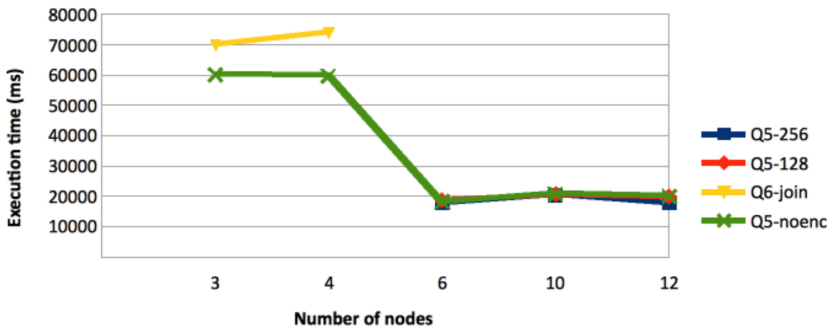


Figure 22: Performance degradation analysis of Query 5 and 6 for 1.5 million entries data set

5.3 Private Cloud Analysis

This section presents and examines experimental results obtained by applying the solution methods for private cloud deployments with AC mechanisms with the general sizing model discussed in problem (P2), see Section D.4.

In order to evaluate the AC, the Multidimensional Knapsack (P4), and the Bin Packing (P5) models, which are introduced in Section D.5, are taken into account. Experiments investigate in particular the MapReduce and Spark Technologies.

Section 5.3.1 describes how the cost of VMs deployed in private clusters has been estimated while the evaluation of job rejections penalties is considered in Section 5.3.2. In Section 5.3.3 two realistic

Copyright © 2017, DICE consortium — All rights reserved

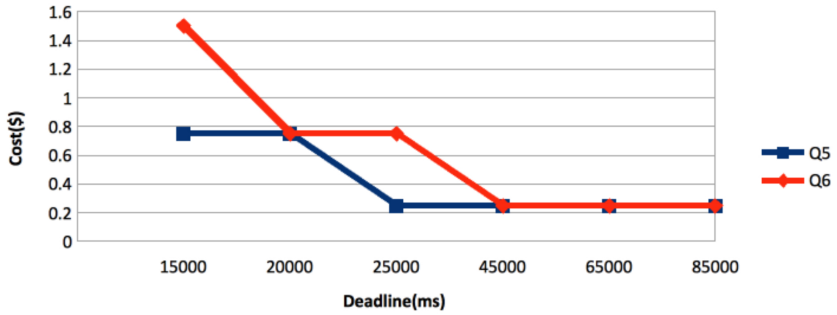


Figure 23: Cost ratio evaluation of Query 5 to 6 by varying deadlines for 1.5 million dataset

scenarios are presented and evaluated by varying the application deadlines, concurrency level, and the cluster size. Section 5.3.4 sheds light on the different results obtained by using the Multiple Knapsack or the Bin Packing model. Finally, Section 5.3.5 provides the scalability analysis comparing the execution time required by the use of Knapsack and Bin Packing models. Scalability analysis is performed only for the private cloud scenario, since this corresponds to the most demanding setting for the **D-SPACE4Cloud** tool. Note that, if $|\mathcal{H}_i|$ denotes the different number of concurrency levels set by the end user for a given class i, a public cloud analysis involves the execution of $|\mathcal{H}_i|$ public cloud instances.

5.3.1 VM Costs

As discussed in Section D.6, the general model (P2) is exploited in the private cloud case considering only δ_j, the unit hourly cost for on demand VMs of type j. This parameter takes into account three main contributions: the energy cost related to the operation of the physical servers hosting the VMs, the overhead energy cost due to cluster maintenance, mainly related to server rooms cooling, and the price of physical servers. The unit energy cost considers European energy prices.[10] Furthermore, the power consumption is estimated taking into account reference benchmarks for servers in production environment, precisely SPECpower.[11] With those benchmarks, the cost of one hour of computation of a single core, $\mathcal{E}$, is estimated. The cooling overhead is modeled applying a multiplier, the *power usage effectiveness* (PUE),[12] to the unit cost of energy. This coefficient is a metric to quantify the relevance of costs that are not directly imputable to the IT equipment in a data center, like lighting, cooling, etc. The price of servers is associated to the modeled computational units with a straight line depreciation method, dividing prices of representative servers, as can be found on manufacturers web sites, by the number of hours in the service life, assumed to be of four years, and the number of physical cores, obtaining the unit cost $\mathcal{S}$.

After estimating the three mentioned contributes, they are aggregated in δ_i. Let d be the server density defined as the virtual to physical core ratio. Moreover, we identify the number of virtual cores per instance with v_i. With all these data, the formula for δ_i is given by:

$$\delta_i = (\text{PUE} \cdot \mathcal{E} + \mathcal{S}) \frac{v_i}{d} \tag{4}$$

In the considered experiments the ratio of allocation of virtual cores to physical cores is $d = 1$. Indeed, since the tasks are CPU intensive, allocating more virtual cores than the available physical ones would degrade the performance.

[10] http://ec.europa.eu/eurostat/statistics-explained/index.php/Energy_price_statistics

[11] https://www.spec.org/power_ssj2008/

[12] http://www.greenpeace.org/international/Global/international/publications/climate/2012/iCoal/HowCleanisYourCloud.pdf

Copyright © 2017, DICE consortium — All rights reserved

5.3.2 Penalties

This section presents a set of experiments that has been performed varying the deadlines and the concurrency level in order to calculate the penalties of each considered class. Through the obtained results the reader can get an idea on how the system reacts to changes in these two aspects. The price retrieved with each analysis has been *scaled* in order to express the *hourly cost paid for sustaining the load due to a single user*. Thus, the penalty of each class is reasonably set an order of magnitude greater than the average of its *scaled* costs. Hence we take the penalty p_i, associated to the rejection of one job in class i, as 10 times the average *scaled* cost.

Even if the charts of this section are relative to some basic cases, they are useful to introduce the reader on how the prediction of the cluster expenses varies by varying some parameters settings. They show the cost variations of a cluster with a fixed capacity N set to a large value in order to avoid job rejections.

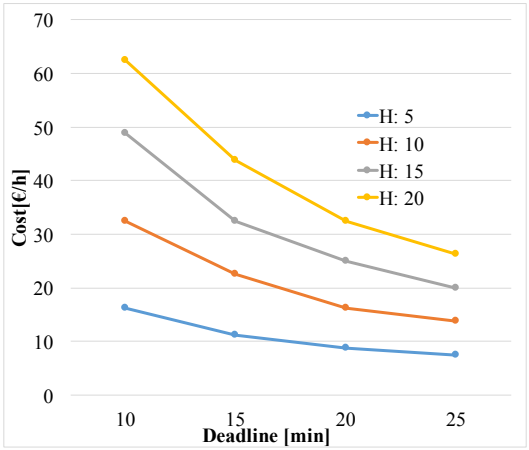


Figure 24: R1, increasing concurrency level w.r.t different deadlines D, no job rejections

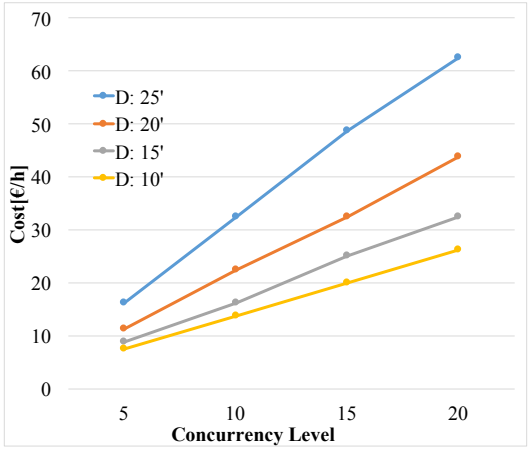


Figure 25: R1, increasing deadlines w.r.t different concurrency levels H, no job rejections

Copyright © 2017, DICE consortium — All rights reserved

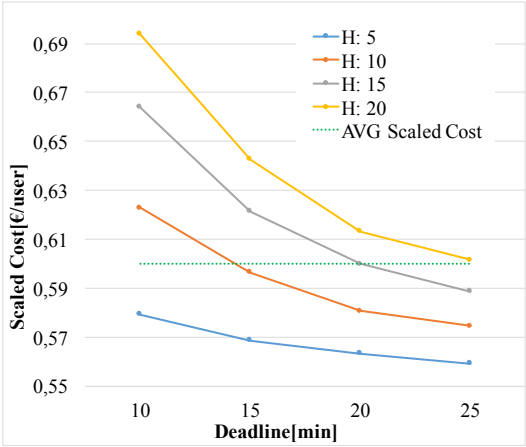


Figure 26: R1 scaled costs

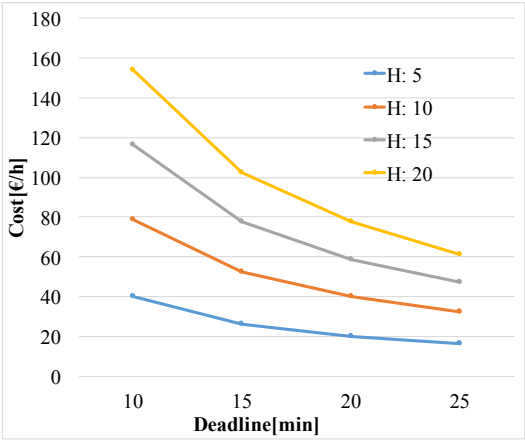


Figure 27: R3, increasing concurrency level w.r.t different deadlines D, no job rejections

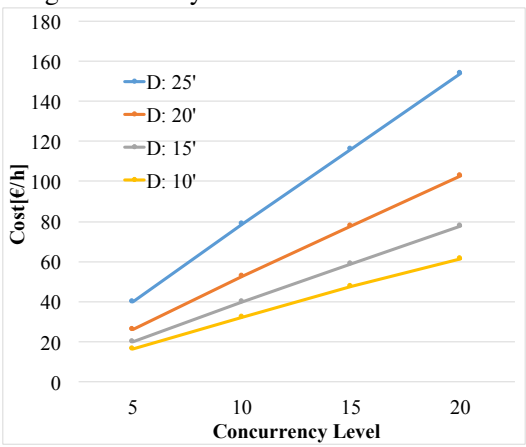


Figure 28: R3, increasing deadlines w.r.t different concurrency levels H, no job rejections

Copyright © 2017, DICE consortium — All rights reserved

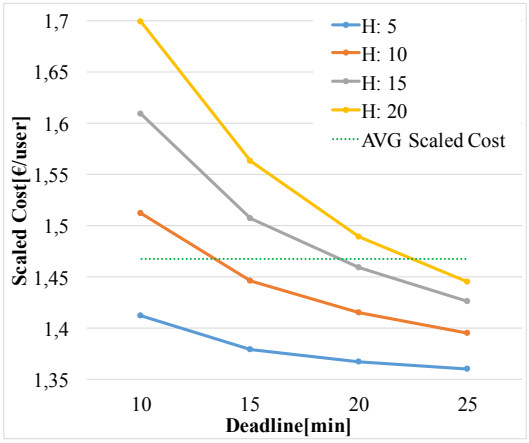


Figure 29: R3 scaled costs

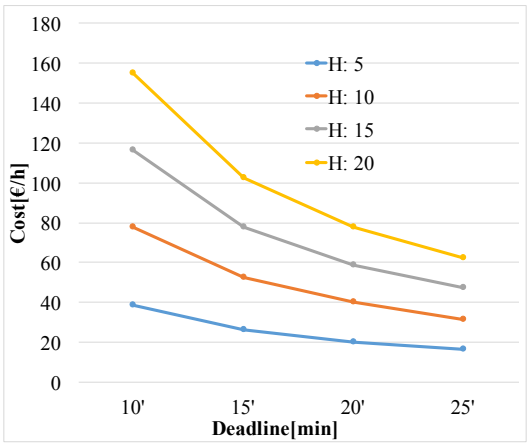


Figure 30: R4, increasing concurrency level w.r.t different deadlines D, no job rejections

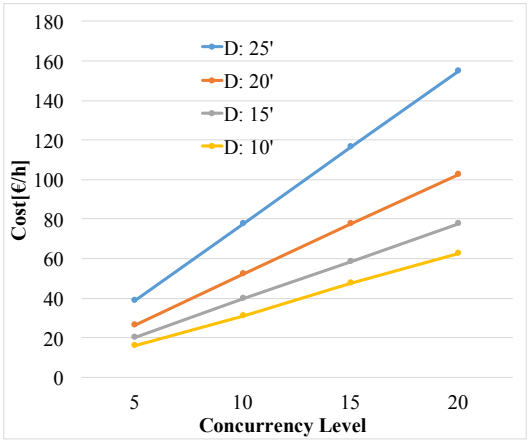


Figure 31: R4, increasing deadlines w.r.t different concurrency levels H, no job rejections

Copyright © 2017, DICE consortium — All rights reserved

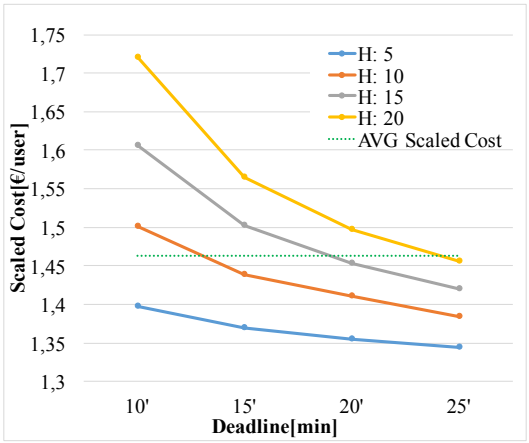


Figure 32: R4 scaled costs

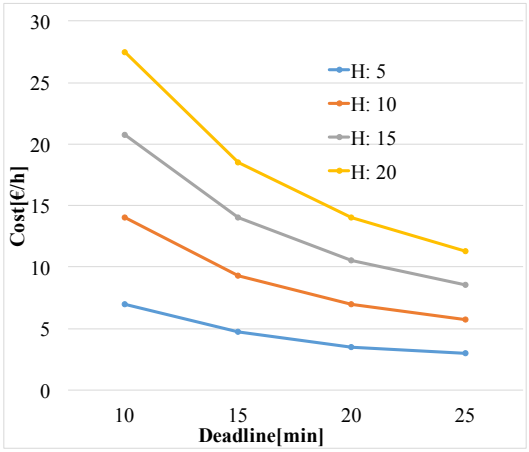


Figure 33: Q1, increasing concurrency level w.r.t different deadlines D, no job rejections

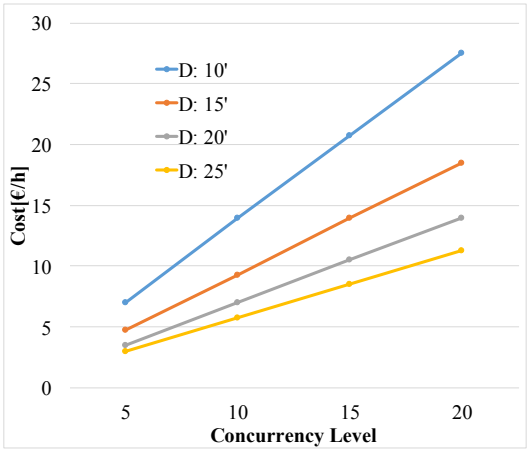


Figure 34: Q1, increasing deadlines w.r.t different concurrency levels H, no job rejections

Copyright © 2017, DICE consortium — All rights reserved

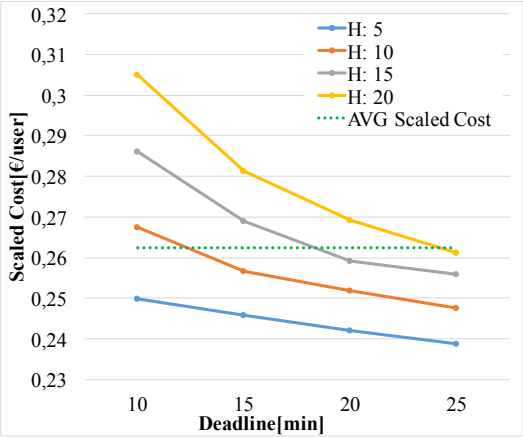


Figure 35: Q1 scaled costs

Copyright © 2017, DICE consortium — All rights reserved

For each class we have reported its query and the graphs used to determine the penalty p_i. Increasing deadlines showed a cost reduction since each job requires less resources of the cluster to satisfy the constraint, this is due to the fact that we can gradually switch off more VMs and consequently physical servers. On the contrary, increasing the number of concurrent users resulted in a cost growth. It is worth to notice that R3 and R4 compared to R1 and Q1 have a significantly higher price. This is in line with the expected results because R3 and R4 work on a larger dataset, so they have an higher cost because they are more computational intensive.

Following the same reasoning, even though Q1 and R1 share the same query, since Q1 has to fetch 30GB of data instead of 148GB, it costs less than R1.

Recall that the *scaled* cost, used to determine the penalties, is the average of the total cost measured per hour divided for the number of users. Hence, the resulting penalties for R1,R3, and R4 are respectively of 6,14,15 $[e/job]$.

Finally it is relevant to highlight that, if not differently specified, the deadlines and the concurrency levels are the same for all the classes.

5.3.3 Scenario-based Experiments

In this section we discuss the analyses performed on two different scenarios to verify that our formulations exhibit behaviors we intuitively expect from the modeled applications. The experiments have been run considering a *medium* case scenario with a cluster composed by $N = 150$ physical machines and a *large* case with $N = 300$ physical machines.

Every studied instance mixes MapReduce and Spark DIAs. Specifically, the instances in our experiments are composed by five classes, i.e., R1, R3, R4 and two Q1 (denoted as Q1 and Q2), characterized by the parameters described in Section 5.1 and the previous sections with different deadlines and concurrent users.

In Big Data environments three main axes govern performance: namely, concurrency level, resource capacity, and deadlines. Isolating each of them allows to understand how the system will react to changes in one single aspect. Hence, we have evaluated the changes in performance in the *medium* case and in the *large* one. For both scenarios we have isolated and gradually incremented application deadlines, concurrency levels and the number of servers in the cluster.

Since we want to study the behavior of the experiments mainly in the AC region, the workloads have to saturate the cluster capacity. For this reason in the *large* case tighter deadlines and larger concurrency levels are considered with respect to the *medium* one.

Finally, notice that the analyses of this section have been performed by only exploiting the Multi Dimensional Knapsack (P4) model, since Bin Packing model solutions took too long. Multi Dimensional Knapsack and Bin Packing model comparison is discussed in Section 5.3.4.

Increasing Deadlines

In this section we fix the cluster capacity, N, and the job concurrency level, H_i^{up} while we increase deadlines and make them less tight at each step. In this case, the predicted costs show a nonlinear behavior. In the beginning, deadlines should be so strict in relation to available resources that they do not allow even minimal operation, hence the problem turns out to be infeasible. Then, the deadlines become less tight and we expect a phase with significant reduction of costs, as jobs do not have to be rejected and penalties to be paid anymore. In the end, when the AC region is left and no penalties are paid, the deadline increase leads to a reduction of the running costs only due to the allocation of a lower number of VMs. Hence, increasing deadlines after the AC region should show a less relevant cost contraction.

Copyright © 2017, DICE consortium — All rights reserved

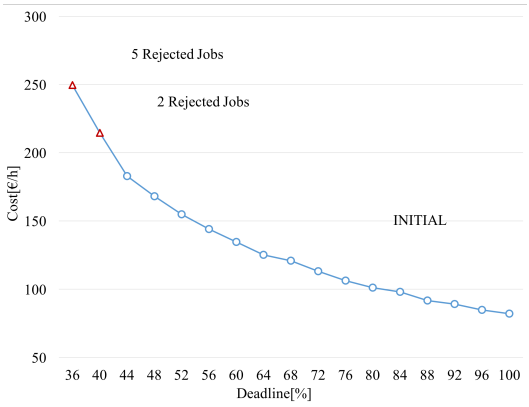


Figure 36: *Medium* case increasing deadlines

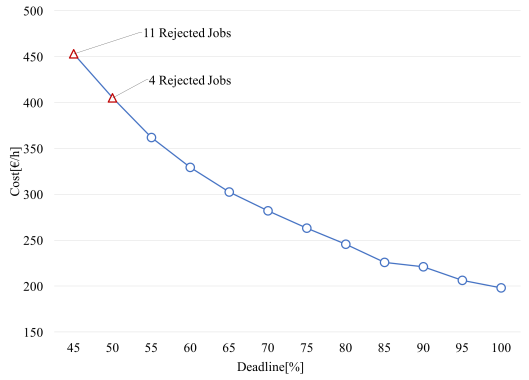


Figure 37: *Large* case increasing deadlines

Figure 36 highlights a great accordance with the expected behavior. In Figure 37 the penalties seem to have no cost impact with respect to the *medium* case, but this behavior is due to the fact that the *large* case starts to reject jobs in a point of the plot where the the slope is larger. In particular, the first feasible instance of this experiment has penalties of $78e$/h and the cluster total cost is of $453e$/h, while the first feasible instance of the *medium* case has $64e$/h and the cluster overall cost is of $249.50e$/h. Hence, in the *large* case the penalties increased the machines cost of 20% while for the *medium* case this increment is of 35%.

Increasing Concurrency Level

In the following, we fix the cluster capacity N and every deadline D_i , then we can predict the behavior under increasing concurrency level, H_i^{up}. For each class of the application the minimum concurrency level has been set to $H_i^{low} = 0.8H_i^{up}$. Indeed, before saturating the capacity we expect a linear increment of costs due an increased requirement of resources, so some VMs and physical servers have to be gradually switched on. After hitting the total capacity, the cluster is not capable of satisfying any additional request, hence we see at first a spike of execution costs due to the penalties associated to job rejections, then the problem becomes infeasible, since the capacity constraint turns out to be incompatible with the bare minimum concurrency level.

Copyright © 2017, DICE consortium — All rights reserved

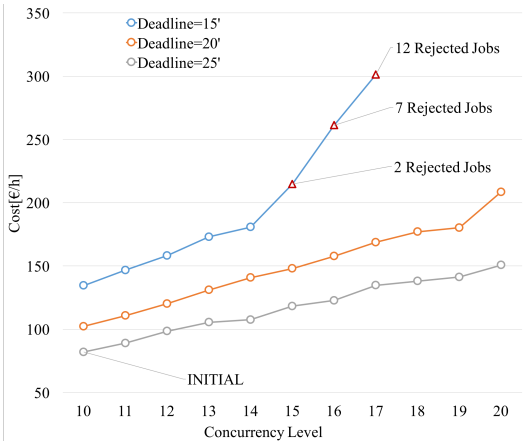


Figure 38: *Medium* case increasing concurrency level with 3 different deadlines

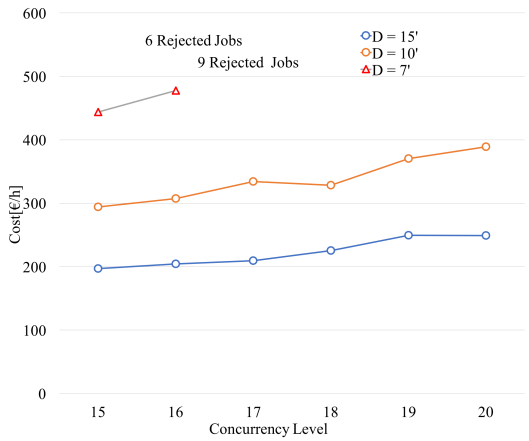


Figure 39: *Large* case increasing concurrency level with 3 different deadlines

Both the Figure 38 and the Figure 39 show that the obtained results are congruent to the expected ones. As before, the penalties contribution in the costs is less evident on the *large* case.

Increasing Capacity

Fixing the maximum concurrency levels H_i^{up} and the deadlines D_i, the parameter that varies in this experiment is the cluster capacity N. Now, we expect to see, after the region where the problem is infeasible, a part of the plot in which some jobs are rejected, thus leading to penalties. Afterwards, when the cluster size is large enough to accommodate all the maximum concurrency levels, so users are not rejected, the predicted cost remains constant even by increasing the number of physical machines. Moreover, we expect that the total number of rejections is not proportional to the cost. Specifically, the cost of a cluster, by increasing capacity, is monotonically nonincreasing. This does not apply when considering the number of rejections.

Copyright © 2017, DICE consortium — All rights reserved

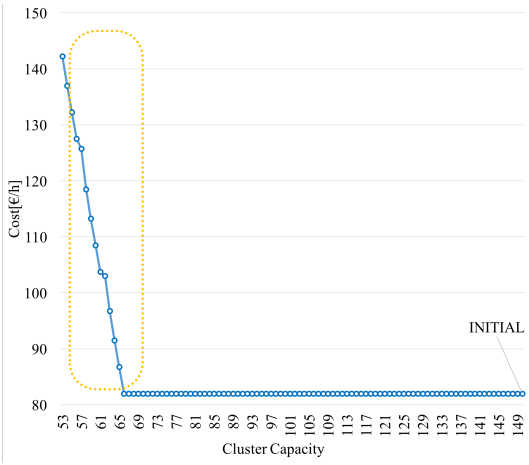


Figure 40: *Medium* case increasing cluster capacity

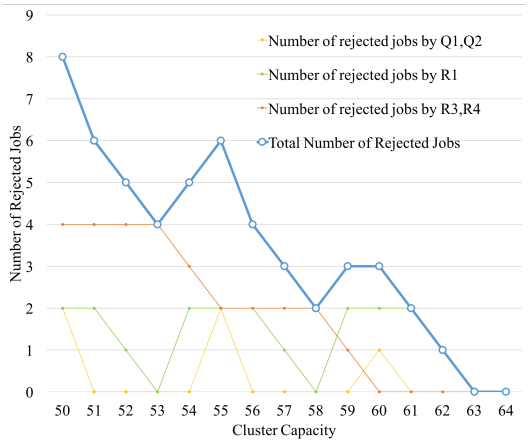


Figure 41: Number of rejection in *medium* case increasing cluster capacity

Copyright © 2017, DICE consortium — All rights reserved

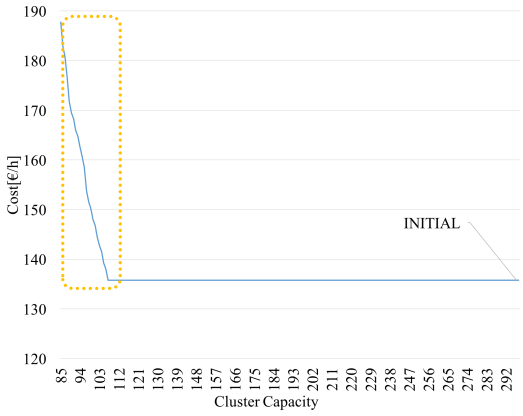


Figure 42: *Large* case increasing cluster capacity

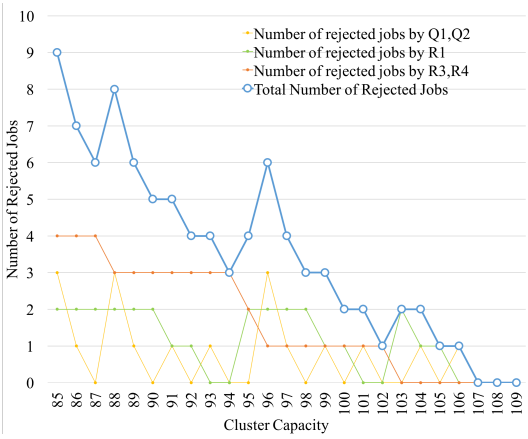


Figure 43: Number of rejections in *large* case increasing cluster capacity

All the plots showed a perfect accordance with the expected behavior. Furthermore, it is worth to notice that in Figure 41 and in Figure 43, the sum of number of rejections of R3 and R4 have a monotonically nonincreasing cost. Indeed, by considering a decreasing capacity of the cluster (i.e., moving from right to left in the previous plots), jobs belonging to those two classes are rejected as late as possible, and, once one of them is rejected, it is never taken into account again. This is a logical behavior, since R3 and R4 are the classes with the highest penalties.

Notice that, for each class i, $|\mathcal{H}_i| = H_i^{up} - H_i^{low} + 1$ expresses the number of different concurrency levels that can be considered. Recall that $H_i^{low} = 0.8H_i^{up}$ and the *large* case has an higher H_i^{up}, in order to hit the cluster capacity. Hence, the $|\mathcal{H}_i|$ is greater than in the *medium* case scenario. This is the reason why, in this subsection, the AC region of the *large* case is wider than in the *medium* case.

5.3.4 Comparisons of Multidimensional Knapsack and Bin Packing

The analyses performed in Section 5.3.3 have exploited only the Knapsack model, since Bin Packing models solutions in such huge experiments took to much. Moreover, as previously stated, with the chosen types of VMs the results given by the two models should not differ too much, since each one of the N homogeneous physical server has the same dimension of the VMs exploited by R1,R3,R4.

This section is in charge of validating the two just described assumptions. Specifically, in order to study the differences in cost optimization between Knapsack and Bin Packing, three experiments with an increasing workload have been examined increasing the number of servers of the cluster. The goal is to examine the execution times of both models and the accuracy of Knapsack with respect to Bin Packing.

The results that we expect from the set of experiments of this section should follow the behavior of

Copyright © 2017, DICE consortium — All rights reserved

analyses conduced in paragraph 5.3.3. However, here we are interested in investigating the difference between the Knapsack and the Bin Packing resulting costs.

In each one of the three experiments, we fix the maximum concurrency levels H_i^{up} and the deadlines D_i, so the parameter that varies is the cluster capacity N. The instances of these analyses are composed, as before, by five classes, R1, R3, R4 and two Q1. The values of the relevant parameters in the three experiments are:

- **Small case:** for each class D_i = 25 minutes and H_i^{up} = 10;
- **Medium case:** for each class D_i = 15 minutes and H_i^{up} = 15;
- **Large case:** for each class D_i = 7 minutes, H_i^{up} = 20, in this case H_i^{low} = 10.

Having in mind Table 2, notice that the large case exploits the minimum D_i and the maximum H_i^{up} level that we are considering for the experiments.

In the experiments of this section the goal is to explore the behavior of some instances in the AC region, in order to compare execution times and resulting cost of both models.

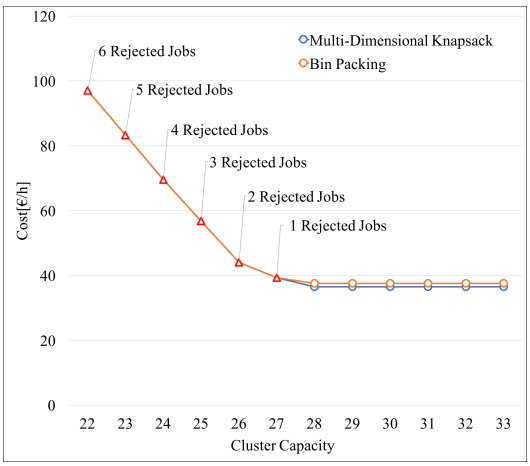


Figure 44: Increasing cluster capacity, small case

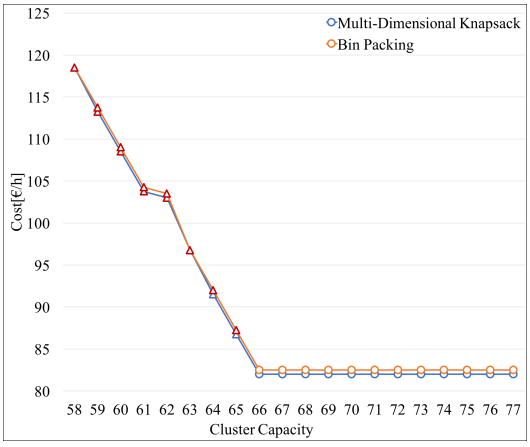


Figure 45: Increasing cluster capacity, medium case

Copyright © 2017, DICE consortium — All rights reserved

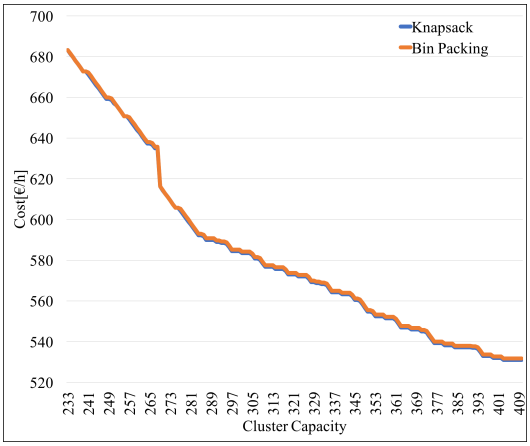


Figure 46: Increasing cluster capacity, large case, variations in costs predictions

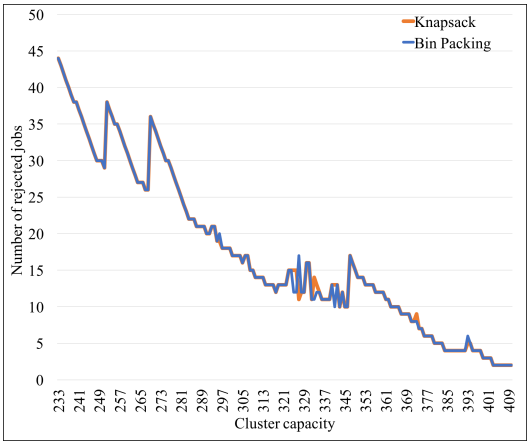


Figure 47: Increasing cluster capacity, large case, variations in the number of rejected users

In Figure 44 both models give the same results in the whole AC region, in terms of cost, number of rejections and type of rejected jobs. Notice that, after the AC region Knapsack returns a lower cost. In Figure 45 and in Figure 46 Knapsack returns a reduced cost with respect to Bin Packing also in the AC region. The results shows that the costs predicted by Bin Packing and by Knapsack are really similar, since the error of Knapsack is lower than 5%.

Since this is an increasing capacity analysis, Figure 44 is reported for conformity with the plots depicted in paragraph 5.3.3. It shows that the fluctuation of rejected jobs does not follow the variation of the costs.

The analyses performed in this section inspect the whole AC region in all the three considered cases, increasing the number of servers with step one. In particular, 12, 20 and 178 runs of Knapsack, first, and Bin Packing, later, have been launched in order to obtain respectively Figure 44, Figure 45 and Figure 46. Figure 47 depicts the difference in the number of rejected jobs between Knapsack and Bin Packing, in case of 178 runs.

The 178 runs required for the large case had a duration of 2 minutes and 33 seconds for Knapsack, whereas for Bin Packing of 80 minutes and 58 seconds (measured as if we are using a single core, and considering only the execution time required to solve the programming problem).

5.3.5 Scalability Analysis

After verifying that our models satisfy some basic properties we intuitively expect, and in line with the other results published in the literature [12], we proceed with a scalability analysis. Our goal is to verify

Copyright © 2017, DICE consortium — All rights reserved

that the proposed approaches allow for solving the optimization problem to support cluster management even in realistic configurations with a large number of different job classes. In particular, we considered an increasing number of classes, i.e., the initial 5 reference classes introduced in the previous sections will be increased with step 5 until an instance composed by 50 classes is studied. Multiple Dimension Knapsack was considered for the AC model.

Table 4 reports the parameters of the five classes added to the initial instance at each step.

Figure 48 shows the time required on a single core as a function of the cluster capacity. Execution time grows linearly as the number of classes. Note that, since **D-SPACE4Cloud** backend was run on a server with 8 cores, the wall time measured in the very worst case was around 31 hours, demonstrating that the parallel implementation is effective and the tool can be used at design time for studying very complex scenarios involving many DIAs.

Table 4: Classes of the Scalability analyses

Step	**Parameter**	**R1**	**R3**	**R4**	**Q1**	**Q2**	**Cluster Capacity**
1	D_i	25	20	20	25	25	50
	H_i^{up}	10	6	6	10	5	
2	D_i	20	20	20	25	25	100
	H_i^{up}	12	6	4	8	5	
3	D_i	10	25	25	25	25	150
	H_i^{up}	5	10	5	5	5	
4	D_i	15	20	25	10	25	200
	H_i^{up}	6	5	5	10	6	
5	D_i	20	25	18	15	25	250
	H_i^{up}	5	5	5	20	5	
6	D_i	25	25	25	20	25	300
	H_i^{up}	5	3	12	10	5	
7	D_i	25	25	25	20	20	350
	H_i^{up}	12	5	5	10	10	
8	D_i	15	25	25	20	25	400
	H_i^{up}	12	3	5	12	4	
9	D_i	25	20	20	20	20	450
	H_i^{up}	10	3	7	10	10	
10	D_i	25	25	25	20	20	500
	H_i^{up}	20	5	4	12	8	

Copyright © 2017, DICE consortium — All rights reserved

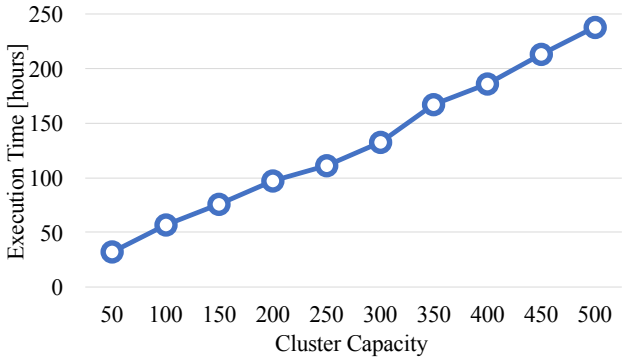


Figure 48: Variation of execution times increasing the capacity and the number of classes on a single core

Copyright © 2017, DICE consortium — All rights reserved

6 Optimizing Container Based Deployments

D-SPACE4Cloud considers as DIAs target deployments private or public cloud virtualized environments. However, the tool can be used to support also container-based deployments. The only assumptions required are that the DTSM models considered as candidate deployments have been profiled in such environments, and that the containers receive performance guarantees both in terms of CPU and memory. If, for example, Docker on Linux is considered this can be easily achieved by setting the *–memory*, *–memory-swappiness*, *–cpus*, *–cpuset-cpus* flags when using the *docker run* command.[13] Thanks to the DICE collaboration with the EUBra-BIGSEA project,[14] which considers the runtime management of DIAs running on Docker containers, the DICE optmimization tool results have been validated also in such contexts. The accuracy of the DICE simulation tools, and hence **D-SPACE4Cloud**, results is very good also in such environments. For example, the deadline predicted by DICE performance models (relying on JMT and GreatSPN) considering the TPC-DS benchmark Q26 running on Spark deployed on three Docker containers with two cores each was 55s against 47s measured on the real system with 17% percentage error, in line with the results discussed in Section 5 and Section 5.2.2.

[13] `https://docs.docker.com/engine/admin/resource_constraints/`
[14] `http://www.eubra-bigsea.eu`

Copyright © 2017, DICE consortium — All rights reserved

7 Conclusions

In this deliverable, we reported the advances in the implementation of **D-SPACE4Cloud** tool achieved between M19 and M30. The tool final version supports MapReduce, Spark and Storm technologies for public and private cloud deployments. The tool has been extensively validated considering industry benchmark applications, the NETF case study, and real cluster deployments. The requirements **R3.8** (Cost/quality balance), **R3.10** (SLA specification and compliance), **R3.9** (Relaxing constraints), **R3IDE.6** (Loading a DDSM level model), see DICE Deliverable D1.2, have been fully achieved. The requirement only missing requirement from D1.2 is **R3.11** (Optimization timeout) which is deprecated, since the time out feature is not always implemented by the underlying simulation frameworks.

Copyright © 2017, DICE consortium — All rights reserved

References

[1] The DICE Consortium. *Requirement Specification*. Tech. rep. European Union's Horizon 2020 research and innovation programme, 2015. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2015/08/D1.2_Requirement-specification.pdf.

[2] The DICE Consortium. *Requirement Specification — Companion Document*. Tech. rep. European Union's Horizon 2020 research and innovation programme, 2015. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2015/08/D1.2_Requirement-specification_Companion.pdf.

[3] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. "JMT: Performance Engineering Tools for System Modeling". In: *SIGMETRICS Perform. Eval. Rev.* 36.4 (2009), pp. 10–15. ISSN: 0163-5999. DOI: 10.1145/1530873.1530877.

[4] Soheib Baarir et al. "The GreatSPN Tool: Recent Enhancements". In: *ACM SIGMETRICS Performance Evaluation Review* 36.4 (2009), pp. 4–9.

[5] Danilo Ardagna et al. "Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets". In: *ICA3PP*. (Granada, Spain). Dec. 2016.

[6] *AMPL*. URL: http://www.ampl.com/ (visited on 07/19/2017).

[7] *CMPL 1.11.0*. URL: http://www.coliop.org (visited on 07/19/2017).

[8] *GLPK (GNU Linear Programming Kit)*. URL: https://www.gnu.org/software/glpk/ (visited on 07/19/2017).

[9] *COIN-OR*. URL: https://www.coin-or.org (visited on 07/19/2017).

[10] Danilo Ardagna, Chiara Francalanci, and Marco Trubian. "Joint Optimization of Hardware and Network Costs for Distributed Computer Systems". In: *IEEE Trans. Systems, Man, and Cybernetics, Part A* 38.2 (2008), pp. 470–484.

[11] Danilo Ardagna et al. *Prediction and Cost Assessment Tool—Final version*. Tech. rep. 2015. URL: http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds_D5.4.3_PredictionAndCostAssessmentToolFinalVersion.pdf.

[12] Marzieh Malekimajd et al. "Optimal Capacity Allocation for Executing MapReduce Jobs in Cloud Systems". In: *Proceedings of the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. (Timisoara, Romania). 2014, pp. 385–392.

[13] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. "ARIA: Automatic Resource Inference and Allocation for MapReduce Environments". In: *Proceedings of the Eighth International Conference on Autonomic Computing*. June 2011.

[14] *Artelys Knitro*. URL: http://www.artelys.com/en/optimization-tools/knitro (visited on 07/19/2017).

[15] Andrey Brito et al. *D3.4 EUBra-BIGSEA QoS infrastructure services intermediate version*. 2017.

[16] *Apache Hive*. URL: https://hive.apache.org (visited on 07/19/2017).

[17] *Apache Spark*. URL: https://spark.apache.org (visited on 07/19/2017).

[18] *TPC-DS Benchmark*. URL: http://www.tpc.org/tpcds/ (visited on 07/19/2017).

[19] *Systems and software engineering — High-level Petri nets — Part 1: Concepts, definitions and graphical notation*. Standard. Geneva, CH: International Organization for Standardization and International Electrotechnical Commission, July 2010.

[20] *Systems and software engineering — High-level Petri nets — Part 2: Transfer format*. Standard. Geneva, CH: International Organization for Standardization and International Electrotechnical Commission, Apr. 2011.

Copyright © 2017, DICE consortium — All rights reserved

[21] *Microsoft HDInsight*. URL: http://azure.microsoft.com/en-us/services/hdinsight/ (visited on 07/19/2017).

[22] *Amazon Elastic MapReduce*. URL: https://aws.amazon.com/elasticmapreduce/ (visited on 07/19/2017).

[23] *The Digital Universe in 2020*. URL: http://idcdocserv.com/1414.

[24] Karthik Kambatla et al. "Trends in Big Data Analytics". In: *Journal of Parallel and Distributed Computing* 74.7 (2014), pp. 2561–2573. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2014.01.003. URL: http://www.sciencedirect.com/science/article/pii/S0743731514000057.

[25] Derrick. "Survey shows huge popularity spike for Apache Spark". In: (2015). URL: http://fortune.com/2015/09/25/apache-spark-survey.

[26] Edward D. Lazowska et al. *Quantitative System Performance. Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984. URL: http://homes.cs.washington.edu/~lazowska/qsp/ (visited on 07/19/2017).

[27] *Amazon EC2 Pricing*. URL: http://aws.amazon.com/ec2/pricing/ (visited on 07/19/2017).

[28] Marzieh Malekimajd et al. "Optimal Map Reduce Job Capacity Allocation in Cloud Systems". In: *SIGMETRICS Perform. Eval. Rev.* 42.4 (June 2015), pp. 51–61. ISSN: 0163-5999. DOI: 10.1145/2788402.2788410.

[29] *Amazon Elastic Compute Cloud (EC2)*. URL: http://aws.amazon.com/ec2 (visited on 07/19/2017).

[30] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

Copyright © 2017, DICE consortium — All rights reserved

A Eclipse plug-in Installation Manual

This appendix provides a user guide for the installation of the **D-SPACE4Cloud** Eclipse plug-in. The description of the frontend and backend components is available in the DICE Deliverable D3.8.

The Eclipse plug-in (the current supported version is Neon available at `http://www.eclipse.org/neon/`) can be installed by performing the following steps:

- Download and run Eclipse
- Select the Help -> Install New Software menu item
- Type `http://dice-project.github.io/DICE-Simulation/updates` as a software site to work with and install all the plugins under this namespace
- Select the Help -> Install New Software menu item
- Select "Neon - http://download.eclipse.org/releases/neon" as a software site to work with
- Expand the Modeling tree item
- Install the UML2 Extender SDK plugin
- Select the Help -> Install New Software menu item
- Select `https://github.com/dice-project/DICE-Optimisation-Plugin` as a software site to work with and install **D-SPACE4Cloud**

Copyright © 2017, DICE consortium — All rights reserved

B Eclipse Plug-in Usage

In this appendix we provide detailed documentation for the **D-SPACE4Cloud** Eclipse plug-in GUI. As discussed in Section 4.1, the plug-in implements a five step wizard whose windows change according to the selected technology and target deployment.

Section 4.1 provides an overview of the plug-in usage when the Spark technology and public cloud deployment are selected. In this section, we report the description of the windows specific for the private deployment and Storm technology.

As discussed in Section 4.1, the optimization wizard start pressing button 1 in Figure 3 or alternatively selecting the entry **Optimization Wizard** from the menu reported Figure 49.

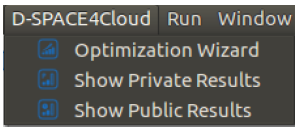


Figure 49: Wizard step 2. Specifying private deployment (physical servers)

In case of private deployment, the end user needs to specify the characteristics of the physical infrastructure available and of the VM types which are in use. In particular **D-SPACE4Cloud** assumes that the physical cluster is homogeneous and allows to specify the characteristics of the physical servers (in terms of number of CPUs, memory available, and cost). Pressing the button *Add new configuration* (see Figure 50) then the user can specify the configuration of the VM types in use. As for the public cloud deployment, multiple configurations can be specified (see Figure 51) and selected as candidate deployment in the following steps of the wizard. In particular, VMs are described in terms of virtual CPUs number, memory and expected hourly costs (which of course depends on the characteristics of the underlying hardware and can be estimated by relying on industry benchmarks [15]).

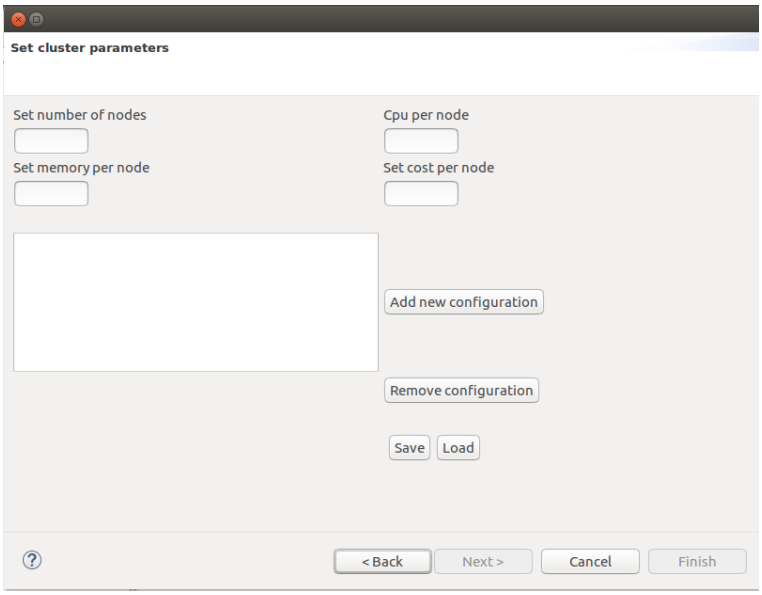


Figure 50: Wizard step 2. Specifying private deployment (physical servers)

Finally, if Storm is selected as DIA reference technology, then the **D-SPACE4Cloud** allows to specify as constraint in step 3 the cluster utilization (a number between 1 and 100) through the window reported in Figure 52.

All the Eclipse plug-in settings (e.g., simulator to be used, front-end and back-end end-points, path of the JMT pre-processor needed to transform PNML files for the JMT simulator, see also Appendix C)

[15] https://www.spec.org/power/

Copyright © 2017, DICE consortium — All rights reserved

can be set through the window shown in Figure 53 which can be accessed through the the Window->Preferences menu selecting afterwards **D-SPACE4Cloud** plug-in.

Copyright © 2017, DICE consortium — All rights reserved

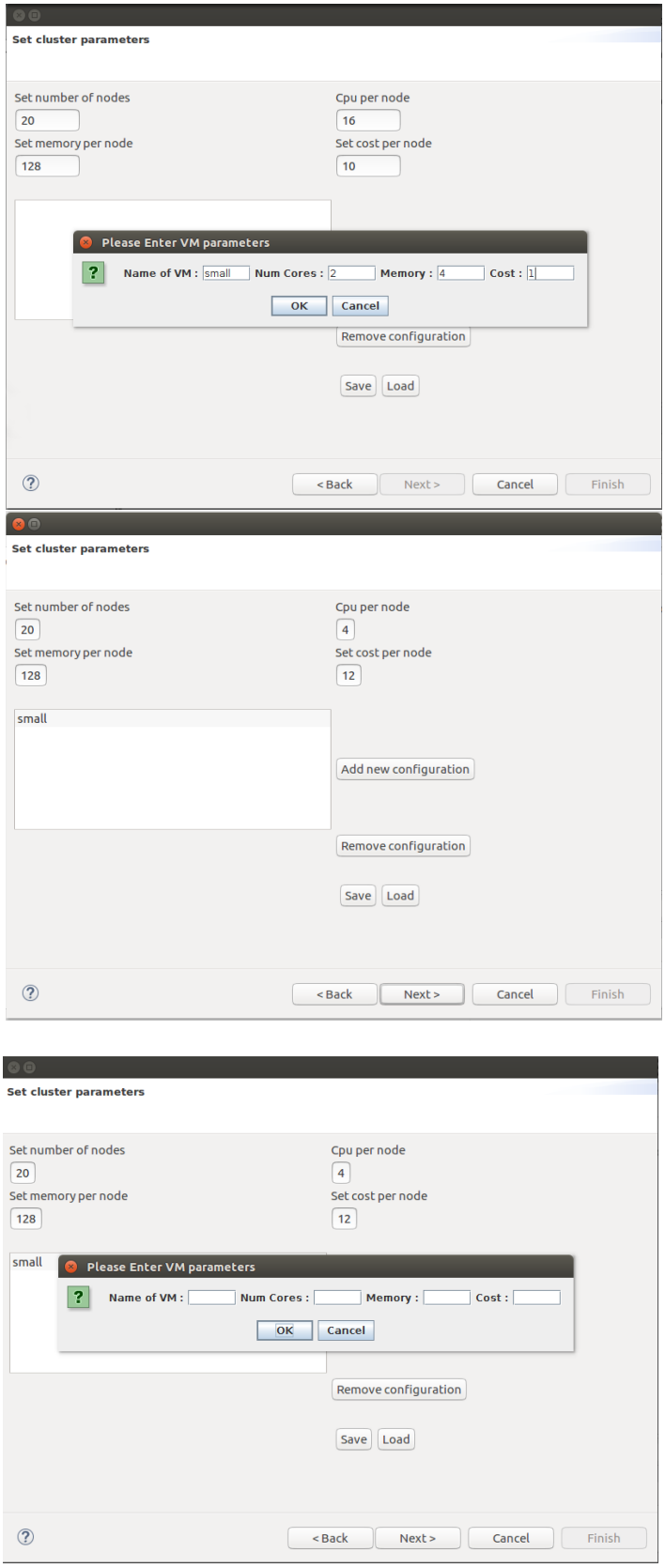


Figure 51: Wizard step 2. Specifying VM type characteristics

Copyright © 2017, DICE consortium — All rights reserved

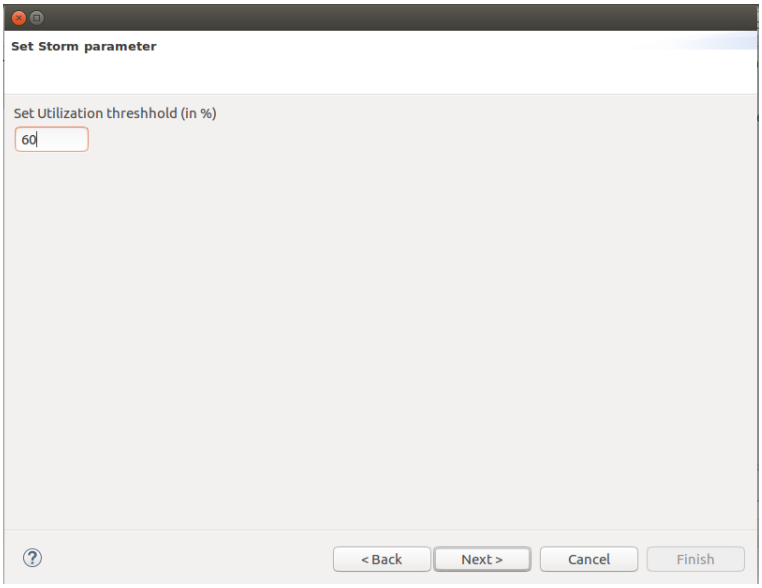


Figure 52: Wizard step 2. Specifying private deployment (physical servers)

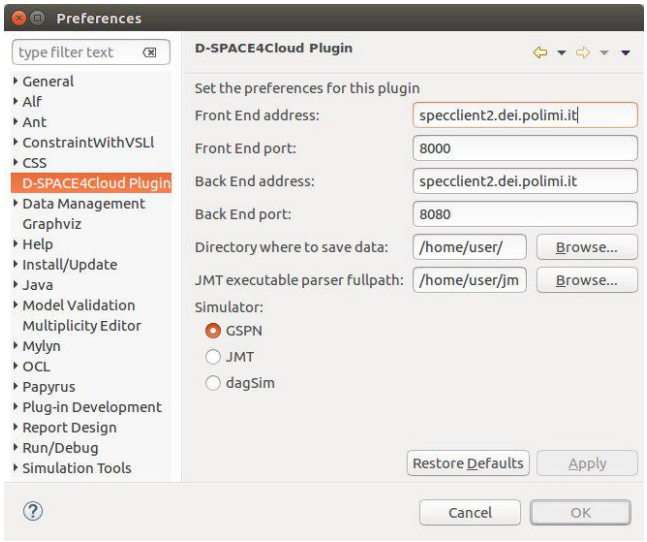


Figure 53: **D-SPACE4Cloud** preferences window

Copyright © 2017, DICE consortium — All rights reserved

C JMT PNML Pre-processor

GSPNs (Generalized Stochastic Petri Nets) and SWNs (Stochastic Well-formed Nets) generated by the M2M transformation mechanism within the DICE Simulation tool are based on the PNML format. PNML (Petri Net Markup Language [16]) is a proposal of an XML-based interchange format for Petri nets standardised by ISO/IEC 15909 ([19] and [20]). The JMT PNML pre-processor [17] is an open-source utility for transforming PNML files into JSIMG files, which can be executed by the JMT simulator. Using the PNML Framework ,[18] the pre-processor keeps compliant and up-to-date with the standard. Besides, it can also identify the tool specific parameters defined by the M2M transformation mechanism.

The ISO/IEC 15909 standard defines four types of PNML models, namely *core models*, *Place/Transition nets*, *symmetric nets* and *high-level Petri nets*. As agreed with the M2M transformation mechanism, the JMT PNML pre-processor pareses PNML files according to the grammar for Place/Transition nets. For GSPNs, JSIMG files simply results from direct interpretation of PNML files. However, since SWNs are not supported by the JMT simulator, only significant characteristics of the models are extracted from PNML files, and JSIMG files are generated by manipulating and parametrizing equivalent hybrid models combining Queuing Networks (QNs) and Colored Petri Nets (CPNs). Figure 54 shows the hybrid model equivalent to an SWN representing a MapReduce application running atop the YARN Capacity Scheduler with a single class of jobs.

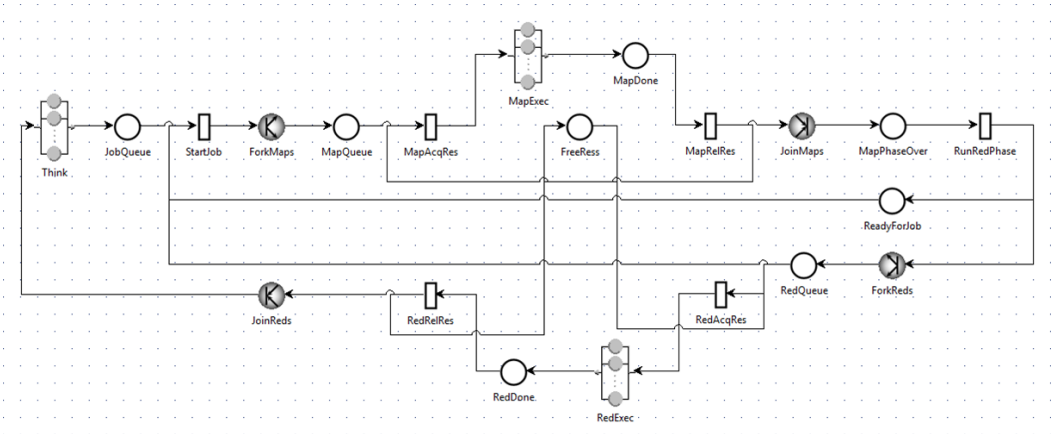


Figure 54: The hybrid model equivalent to an SWN representing a MapReduce application running atop the YARN Capacity Scheduler with a single class of jobs

The grammar for Place/Transition nets does not support all the features of GSPNs and SWNs, for example the temporal specification of a transition. The extra features are covered by identifying the tool specific parameters defined by the M2M transformation mechanism. The tool specific parameters illustrated in Listing 1 are applicable to the *Net* element, allowing the *color set definition* and the *color definition* of SWNs. The color set definition encloses two values specifying the *name* and the *ordering* of the color set. The three values in the color definition correspond to the *ID*, the *name* and the *number of tokens* of the color. With the tool specific parameters illustrated in Listing 2, the *timing semantics* and the *firing time distribution* for the *Timed Transition* element are enabled. The timing semantics is either *infinite-server* or *single-server* depending on whether the first or second tool specific parameter is present. In the current definition, the firing time distribution may only be exponential, and the enclosed value specifies the rate of the distribution. As illustrated in Listing 3, the two tool specific parameters for the *Immediate Transition* element respectively provides the values of the *firing priority* and *firing weight*. Listing 4 illustrates the only tool specific parameter applicable to the Arc element. The presence of this tool specific parameter indicates whether the Arc element represents an inhibitor or normal arc.

[16] `http://www.pnml.org/`
[17] `https://github.com/dice-project/DICE-Optimisation-JMT-Pre-processors`
[18] `http://pnml.lip6.fr/`

Copyright © 2017, DICE consortium — All rights reserved

Listing 1: Tool specific parameters for the Net element

```
<!-- Color Set Definition -->
<toolspecific tool="DICE␣PNML␣Extensions" version="0.1">
    <value grammar="http://es.unizar.dsico/pnconstants/color/colorset">C0</value>
    <value grammar="http://es.unizar.dsico/pnconstants/color/colorset">0</value>
</toolspecific>
<!-- Color Definition -->
<toolspecific tool="DICE␣PNML␣Extensions" version="0.1">
    <value grammar="http://es.unizar.dsico/pnconstants/color/color">0</value>
    <value grammar="http://es.unizar.dsico/pnconstants/color/color">c</value>
    <value grammar="http://es.unizar.dsico/pnconstants/color/color">3</value>
</toolspecific>
```

Listing 2: Tool specific parameters for the Timed Transition element

```
<!-- Infinite-server Semantics -->
<toolspecific tool="DICE␣PNML␣Extensions" version="0.1">
    <value grammar="http://es.unizar.dsico/pnconstants/tserv/infinite"/>
</toolspecific>
<!-- Single-server Semantics -->
<toolspecific tool="DICE␣PNML␣Extensions" version="0.1">
    <value grammar="http://es.unizar.dsico/pnconstants/tserv/one"/>
</toolspecific>
<!-- Firing Time Distribution -->
<toolspecific tool="DICE␣PNML␣Extensions" version="0.1">
    <value grammar="http://es.unizar.disco/pnconstants/tkind/exponential">1.0</value>
</toolspecific>
```

Listing 3: Tool specific parameters for the Immediate Transition element

```
<!-- Firing Priority -->
<toolspecific tool="DICE␣PNML␣Extensions" version="0.1">
    <value grammar="http://es.unizar.disco/pnconstants/tkind/immediatepriority">1</value>
</toolspecific>
<!-- Firing Weight -->
<toolspecific tool="DICE␣PNML␣Extensions" version="0.1">
    <value grammar="http://es.unizar.disco/pnconstants/tkind/immediate">1.0</value>
</toolspecific>
```

Listing 4: Tool specific parameters for the Arc element

```
<!-- Inhibitor Arc -->
<toolspecific tool="DICE␣PNML␣Extensions" version="0.1">
    <value grammar="http://es.unizar.disco/pnconstants/akind/inhibitor"/>
</toolspecific>
```

To deploy the JMT PNML pre-processor within **D-SPACE4Cloud**, download the `PNML_Pre_Processor` subfolder from `https://github.com/dice-project/DICE-Optimisation-JMT-Pre-processors`, and then specify the full path of its local copy through the **D-SPACE4Cloud** preferences window shown in Figure 53.

Copyright © 2017, DICE consortium — All rights reserved

D Resource Provisioning Problems Mathematical Formulation

In this appendix we summarize the main concepts introduced in Section 4 and we present the notation required to formulate the problem of the determining DIAs minimum cost deployment configurations as an optimization problem, considering both private and public cloud environments.

As discussed in Sections 3 and 4.2, several user classes require the execution of DIAs that, according to SLA requirements, need to be completed providing QoS guarantees (e.g., before given deadlines expire, with a minimal concurrency level, i.e., number of concurrent jobs). This problem is also particularly interesting when the available capacity does not allow to allocate the entire submitted workload, thus forcing the rejection of some jobs and the payment of the related penalties. In this scenario, knowledge about the resource requirements of each user has to be exploited to minimize costs.

This appendix starts with Section D.1, introducing the problem statement and explaining the different use case scenarios and modeling assumptions in Section D.2. Afterwards, Section D.3 shows how the execution times can be estimated through machine learning (ML) (ML execution time estimate is exploited to identify an initial solution for the HC algorithm discussed in Section 4.2). Sections D.4 D.5 illustrates the optimization problem from a mathematical point of view, in order to provide a basis for the solution techniques proposed in Section 3. Finally, in Section D.6 a mapping between scenarios described in Section 3.1 and the optimization models is reported. The formulation will focus on Apache MapReduce and Spark technologies. Section D.7 describes the main differences when the DIA is based on Storm.

D.1 Problem Statement

Nowadays, Apache MapReduce and Spark, are among the most widely used solutions to support Big Data applications. Back to the begin of the Big Data era, clusters were deployed in house exploiting the internal infrastructure of companies. Hadoop MapReduce was the technology leader on the market and it was installed mainly on bare metal. Initially, only one DIA can be executed on the underlying cluster, since Hadoop assigned all available resources to that application. In these circumstances, the execution times only depended on the particular dataset. Soon this scenario changed: to improve utilization, resources were virtualized and the cluster shared among different heterogeneous DIAs. Later, in order to avoid large upfront investments and maintenance costs, the tendency was to outsource the resource management and to provision clusters in the cloud. In public clouds, the computing infrastructure was far more vast and scalable than in private clouds. Furthermore, the new paradigm allowed to pay only for the resources that are used without worrying about infrastructure costs and technical or architectural issues. Hence, the pay-per-use approach and the almost infinite capacity of cloud infrastructures started to be used efficiently in supporting data intensive computation. Nowadays, many cloud providers already include in their offering MapReduce and Spark based platforms, among which Microsoft HDInsight [21] or Amazon Elastic MapReduce [22]. IDC estimates that, by 2020, nearly 40% of Big Data analyses will be supported by public clouds [23]. While Hadoop touched half of the world data in 2014 [24] many experts assume that Spark will dominate the Big Data market for the next 5-10 years [25].

Within **D-SPACE4Cloud**, the system ARCHITECT, which executes concurrently DIA or DIAs, with similar performance profile (see Section D.2), has to consider many factors altogether: application topology, cluster composition and size, leasing costs of virtual resources, computational power in house, and expected concurrency levels (i.e., the number of concurrent users). The ARCHITECT specifies non-functional requirements in terms of deadlines for the application along with a set of suitable providers and VM types for each of them. In case of private cloud, a description of the available hardware and VMs must be provided as well.

The optimization process is designed to be carried out in an agile way. More precisely,
D-SPACE4Cloud fosters an approach whereby cluster optimization can be performed through an environment that hides the complexity of the underlying models and of the performance evaluation and optimization engines. This frees the ARCHITECT from being an expert of performance prediction, simulation, and techniques on which the tool is grounded, allowing her/him to focus only on the scenarios of her/his interest.

Copyright © 2017, DICE consortium — All rights reserved

D.2 Modeling Assumptions

This section sheds light on the specific techniques and assumptions underlying the mathematical programming formulation. The focus is on modeling clusters adopting Hadoop 2.x MapReduce, Spark ruuning on top of the YARN Capacity Scheduler.

D-SPACE4Cloud works under the hypothesis that YARN is configured in a way that all the available cores can be dynamically assigned to DIA tasks. The system (see [26]), can be represented as a closed model, where users work interactively submitting new jobs after an exponentially distributed think time (details on the performance models can be found in DICE Deliverable D3.8 and [5]).

In order to obtain the most cost-effective configuration, a catalog of VMs with different characteristics is examined. In case of public cloud, the catalog of VMs is given by Infrastructure as a Service (IaaS) providers. This list of configurations might be possibly large, including different characteristics and costs, making the right design-time optimization a challenge that can lead to important savings throughout the cluster life-cycle. For public clouds a pricing model derived from *Amazon EC2* [27] is considered here. The provider offers: 1) *reserved* VMs, adopting a one-time payment policy and granting access to a certain number of them for the contract duration, either at a strongly discounted hourly fee or without further variable costs; 2) *spot* VMs for which customers bid and compete to exploit unused datacenter capacity, yielding very competitive hourly fees; and 3) *on-demand* VMs, which customers can buy at a higher price to compensate peaks in the incoming workload, but without any long term commitment. To obtain the optimal configuration, it is possible to rely on reserved VMs for the bulk of computational needs and complement them with either spot or on-demand instances (*spot* instances are supported only by the premium release of **D-SPACE4Cloud**). In case of private cloud, the catalog of VM configurations has to be provided by the ARCHITECTs and VM instances with the same configuration will have the same cost.

Different classes gather applications that show a similar behavior. The clusters composition and size, in terms of type and number of VMs, must be decided in such a way that, for every application class i, H_i[19] jobs are guaranteed to execute in parallel and complete before a prearranged deadline D_i.

Note that, the execution of jobs on a suboptimal VM type might give rise to performance disruptions, hence it is critical to avoid assigning tasks to the wrong instance type. Indeed, YARN allows for specifying *Node Labels* and *partitioning nodes* in the cluster according to them, thus enforcing this constraint. The configuration resulting from the optimization statically splits different VM types with this mechanism and adopts within each partition either a further static separation in classes or a work conserving scheduling mode, whereby idle resources can be assigned to jobs requiring the same instance type. The assumption on the scheduling policy governing the exploitation of idle resources is not critical. It only affects the interpretation of results, where the former case leads to sharp predictions, while, in the latter, the outcomes of the optimization algorithm are upper bounds, with possible performance improvements due to a better cluster utilization. In light of the above, is clear that the ultimate goal of the proposed approach is to determine, for all the classes, the optimal VM type, number, and pricing model (in case of public clouds), such that the sum of costs is minimized while the deadlines and concurrency levels are met.

The reader is referred to Figure 55 for a graphical overview of the main elements of the considered reference technology system.

Further considerations have to be assumed in the private cloud case. In this scenario, since the capacity of the cluster is limited, it is not always possible to handle the submitted workload, but job rejection can be exploited to obtain a bearable charge. So, in the private cloud case, pecuniary penalties p_i due to job rejections are considered to tackle some decreases in the concurrency levels H_i^{up} requested by the system ARCHITECT. The ARCHITECT has to specify H_i^{low}, that is a lower bound of the concurrency level for each class. Thus, when AC is taken into account, the ultimate goal, previously introduced, aims also at determining, for each class, the optimal number of concurrent users h_i, with $H_i^{low} \leq h_i \leq H_i^{up}$.

[19] In the following, the number of concurrent jobs of a class, i.e., the concurrency level, will be denoted alternatively with H_i and h_i. In particular, we will adopt H_i when the concurrency level is fixed and hence can be considered a parameter of the optimization problem. h_i will be used when **D-SPACE4Cloud** has to determine the optimal number of concurrent jobs, i.e., h_i is a decision variable of the underlying optimization problem.

Copyright © 2017, DICE consortium — All rights reserved

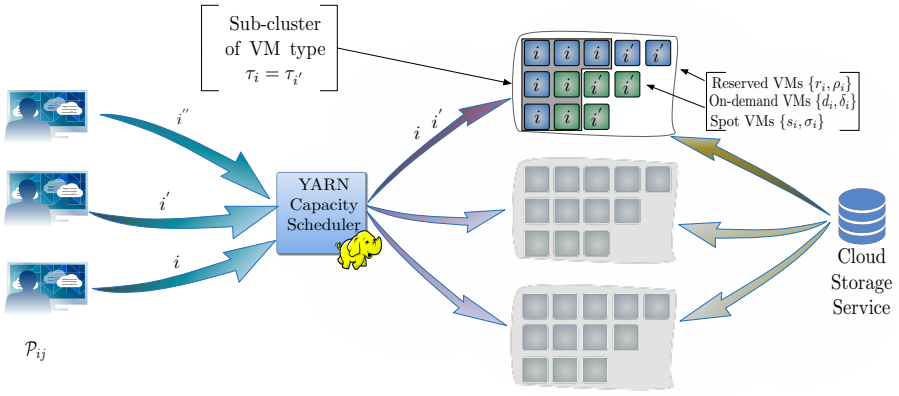


Figure 55: Reference system

We assume that the private cloud cluster is composed by N homogeneous servers, each one with a given amount of memory M and a given number of CPU V.

Note that, the hourly cost of each in house VMs includes:

- cost due to servers provisioning;
- facility costs, such as management costs, employees payments and data center maintenance costs;
- pecuniary job rejection penalties;
- cost of electricity to power servers and cooling systems.

The work performed in DICE proposes two different models to support the private cloud with jobs rejections, i.e., Admission Control (AC), see Section 3.1, where:

- data centers are modeled as a continuum. It is the most efficient model among the two, but it tries to reduce the overall cost without considering how the VMs can be deployed in the given physical servers (see Section D.5.1).
- also servers of the data center are modeled. This model minimizes the cost optimizing also the distribution of the VMs among the physical nodes, allowing potential savings and adding constraints not considered in the first model (see Section D.5.2).

D.3 Predicting Execution Times through a Machine Learning Technique

D-SPACE4Cloud proposes a Hill Climbing (HC) approach to resolve the capacity allocation problem introduced in Section D.1 and Section D.2. In particular, this section provides the description of the machine learning technique used in the initial phase of the HC algorithm. First, general motivations behind this technique choice are presented, then its formulation will be shown.

Nowadays, many works proposed analytical models used to describe the behavior of Big Data clusters. This tendency underlines that building an accurate performance model, able to estimate job execution time of MapReduce or Spark applications, is a big challenge. The recent improvements introduced by Hadoop 2.x in dynamically handling containers among DIA tasks, along with improving the overall cluster utilization, increased further the complexity of those models. So, analytical models have two main problems: 1) they are time consuming; 2) a deep understanding of the underlying environment is required to build them, thus, they are very sensitive on changes of the modeled technology. In order to reduce the prediction times and to free the models from the details of the employed technology,

Copyright © 2017, DICE consortium — All rights reserved

this section summarizes a technique that overcomes those problems that has been developed within the EUBra-BIGSEA project (see EUBra-BIGSEA Deliverable D3.4 [15]) and has been integrated within **D-SPACE4Cloud** premium version.

The chosen machine learning technique is Support Vector Regression (SVR), which is intrinsically more robust than other techniques, such as neural networks, and less sensitive to outliers in the training set than, for example, linear regression. In addition, it can scale to several dimensions. The overall idea is that DIAs execution time depends on a complex relation among some features describing the DIA and the cluster. By using the SVR machine learning technique, it is possible to obtain an approximation of such relation. SVR extracts from a set of experiments the relation between a DIA execution time and some of its features, rather than manually building such relation. The extracted relation can be exploited on DIAs that are different from the ones in the training set just by using their features.

The goal of this approach is to predict the performance of DIAs constituted by a DAG of stages. As described in Section 4, **D-SPACE4Cloud** exploits a combination of the analytical models introduced in DICE Deliverable D3.4 and this machine learning technique to provide a promising initial solution as soon as possible. The analytical model are used as little as possible, since they are much slower than SVR. Hence, this approach, is used in order to reduce the space of predictions to be analyzed with analytical models.

Section D.3.1 gives a general formulation of the SVR and introduces the features exploited by **D-SPACE4Cloud**.

D.3.1 SVR Formulation

Given the feature vector $\bar{x}_i \in \mathbb{R}^n$ and the target output $y_i \in \mathbb{R}$, a set of m training point can be expressed as $\{(\bar{x}_1, y_1), ..., (\bar{x}_m, y_m)\}$. Having this definition in mind, the final goal of SVR can be described as the calculation of the "flattest" linear function $f(\bar{x}) : \mathbb{R}^n \to \mathbb{R}$, which approximates every point in the training set with an error lower that ε, i.e., $\forall i \in [1, m] \cdot |y_i - f(\bar{x}_i)| < \varepsilon$. Since $f(\bar{x})$ is linear, $f(\bar{x}) = \bar{w}^T \bar{x} + b$, thus the flatness of $f(\bar{x})$ depends on vector $\bar{w}$ being small. So it is possible to search $f(\bar{x})$ as the linear function having the smallest norm of $\bar{w}$, satisfying the constraint on the maximum error ε, obtaining the following optimization problem:

$$\min_{\bar{\boldsymbol{w}},\boldsymbol{b}} \quad \frac{1}{2}\bar{w}^T\bar{w}$$

subject to:

$$y_i - \bar{w}^T\bar{x}_i - b \le \varepsilon$$
$$\bar{w}^T\bar{x}_i + b - y_i \le \varepsilon$$

Since function $f(\bar{x})$ could not exists if ε is too small, it is possible to relax the constraint over ε, obtaining the following problem:

$$\min_{\bar{\boldsymbol{w}},\boldsymbol{b},\bar{\boldsymbol{\xi}},\bar{\boldsymbol{\xi}}^{\boldsymbol{*}}} \quad \frac{1}{2}\bar{w}^T\bar{w} + C\sum_{i=1}^{m}(\xi_i + \xi_i^*)$$

subject to:

$$y_i - \bar{w}^T\bar{x}_i - b \le \varepsilon + \xi_i$$
$$\bar{w}^T\bar{x}_i + b - y_i \le \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \le 0$$

Copyright © 2017, DICE consortium — All rights reserved

In practice the slack variables ξ_i, ξ_i^* are added to relax the constraint on the maximum allowed error. It is possible to tune error sensitivity through parameter C: the smaller is C the more errors are ignored; on the contrary the bigger is C the more the deviations from the points above ε are considered.

The choice of the features to be used to predict the DIA durations has been based on previous works as [13], [28], and EUBra-BIGSEA Deliverable D3.4 [15]. These works presented models able to predict MapReduce job execution time, based on the number of map and reduce tasks, the average and maximum duration of map, shuffle and reduce tasks and the available number of CPU cores. Moreover, in addition to these features justified by models presented in literature, the predictions in **D-SPACE4Cloud** are based also on other features. In particular, some of them are based on measures of the amount of data involved in the DIA, such as a descriptor of the data size involved and the average and maximum number of bytes transferred during shuffles.

The execution time of a DIA is directly proportional to all those features, except to the number of CPU cores (*numCores*). In fact job execution time is inversely proportional to the number of cores. Thus, in order to remove this non-linear dependency, the reciprocal of the number of cores is considered ($\frac{1}{numCores}$) as a feature.

All the features discussed up to now can be applied for pure two stage MapReduce jobs. Regarding Spark DIAs, features include the computation and shuffle average and maximum time of the tasks for each DAG stage.

D.4 Problem Formulation

In Section D.1 the optimization problem has been introduced, specifically Section D.2 illustrates the assumptions necessary to underestand the mathematical formulation presented in this section. First of all, to provide an initial general formulation, the focus is on the public cloud case, considering three types of purchasing options, namely reserved, on demand and spot instances. Afterwards, it is shown how the given formulation can be adapted to fit also the private case.

The Hadoop framework features a Resource Manager (RM), entitled to split the available resources among the submitted jobs. These resources are then exploited for computation by the Application Masters (AMs): every AM represents a job class and is the entity that asks for resources and launches the tasks needed to carry out the jobs in its job class (see also DICE Deliverable D3.4). Different classes $\mathcal{A} = \{i \,|\, i = 1, \ldots, n\}$ gather applications that show a similar behavior.

The cluster composition and size, in terms of type and number of VMs, must be decided in such a way that, for every application class i, H_i jobs are guaranteed to execute concurrently and complete before a prearranged deadline D_i.

IaaS providers feature a catalog of VM configurations $\mathcal{V} = \{j \,|\, j = 1, \ldots, m\}$ that differ in capacity (CPU speed, number of cores, available memory, etc.) and cost. Making the right design-time decision poses a challenge that can lead to important savings throughout the cluster life-cycle.

From now on r_i, d_i and s_i are considered as the number of, respectively, reserved, on-demand and spot VMs associated to class i. ν_i is the total number of leased VMs, it can be expressed as $\nu_i = r_i + d_i + s_i$. σ_j is the unit cost for spot VM of type j, whilst δ_j is the unit cost for on-demand VMs, and ρ_j is the effective hourly cost for one reserved VM (it is evaluated on the unit upfront payment normalized over the contract duration). See [29] for more details about the considered pricing models. Denoting with τ_i the type of the VM associated to class i, the cluster hourly cost is:

$$\mathcal{C} = \sum_{i \in \mathcal{A}} \left(\sigma_{\tau_i} s_i + \delta_{\tau_i} d_i + \rho_{\tau_i} r_i \right) \tag{5}$$

As discussed Section 4.1 end users bid and compete for spot VMs and when the spot price is higher than user's bid, the IaaS provider can terminate spot instances without notice. For this reason, the reliability of spot VMs depends on market fluctuations and to keep a high QoS the number of spot VM is bounded not to be greater than a fraction η_i of ν_i for each class i.

Copyright © 2017, DICE consortium — All rights reserved

Table 5: Model parameters

Parameter	Definition
$\mathcal{A}$	Set of application classes
$\mathcal{V}$	Set of VM types
h_i	Number of concurrent users for class i
Z_i	Class i think time [ms]
D_i	Deadline associated to applications of class i [ms]
R_i	Maximum number of reserved VMs allowed to class i
η_i	Maximum percentage of spot VMs allowed to class i
σ_j	Unit hourly cost for spot VMs of type j [€/h]
δ_j	Unit hourly cost for on demand VMs of type j [€/h]
ρ_j	Effective hourly price for reserved VMs of type j [€/h]
$\mathcal{P}_{ij}$	Job profile of class i with respect to VM type j
m_i	Container size in RAM associated to applications of class i [GB]
v_i	Number of vCPUs associated to applications of class i
M_i	Total memory available per Node Manager associated to applications of class i [GB]
V_i	Total number of vCPUs available per Node Manager associated to applications of class i

Moreover, to correctly model and solve this problem, it is crucial to predict, with fair confidence, the execution times of each application class under different conditions: level of concurrency, cluster size, and composition. Following the notation brought forth in [13, 12], given a certain VM of type j, the job profile $\mathcal{P}_{ij}$ for application class i aggregates the features described in Section D.3.

Given the amount and type τ_i of resources allocated, the concurrency level, and the job profile, the estimated execution time can generically be expressed as:

$$T_i = \mathcal{T}\left(\mathcal{P}_{i,\tau_i}, \nu_i, H_i; Z_i\right), \quad \forall i \in \mathcal{A}. \tag{6}$$

Note that, the previous formula represents a general relation describing either closed form results, as those presented in [12] and Section D.3.1 or the average execution times derived via simulation, the approach adopted in **D-SPACE4Cloud** to identify the final optimal solution.

Equations (6) can be used to formulate the deadline constraints as:

$$T_i \leq D_i, \quad \forall i \in \mathcal{A}. \tag{7}$$

With this informations, it is possible to say that **the ultimate goal of the proposed approach is to determine the optimal VM type selection τ_i and number and pricing models of VM $\nu_i = r_i + d_i + s_i$ for each class i such that the sum of costs is minimized while the deadlines and concurrency levels are met**.

The reader is referred to Figure 55 for a graphical overview of the main elements of the considered resource provisioning problem in relation to the reference technologies. Furthermore, in Table 5 a complete list of the parameters used in the models presented in the next sections is reported, whilst Table 6 summarizes the decision variables.

In the following, the optimization models and techniques exploited by the **D-SPACE4Cloud** tool are presented. These approaches are used in order to determine the optimal VMs mix, given the profiles characterizing the applications under study and the possible cloud provider to host the virtual cluster.

The models of the system under study are the basic building blocks for the optimization tool. First of all, a mathematical programming formulation is exploited to estimate the completion times and operational costs. With this formulation, it is possible to swiftly explore several possible configurations and point out the most cost-effective among the feasible ones. Afterwards, the required resource configuration can be fine-tuned using more accurate, even if more time consuming and computationally

Copyright © 2017, DICE consortium — All rights reserved

Table 6: Decision variables

Variable	Definition
x_{ij}	Binary variable equal to 1 if class i is hosted on VM type j
ν_i	Number of VMs assigned for the execution of applications from class i
r_i	Number of reserved VMs booked for the execution of applications from class i
d_i	Number of on-demand VMs assigned for the execution of applications from class i
s_i	Number of spot VMs assigned for the execution of applications from class i
c_i	Number of containers allocated to class i

demanding simulations (thorough JMT, GreatSPN or dagSIM with the premium release), reaching an accurate prediction of the expected response time.

According to the previous considerations, the first step in the optimization procedure consists in determining the most cost-effective resource type, based on price and expected performance. This will be done by exploiting a set of logical variables x_{ij}. It is important to underline that $x_{i,\tau_i} = 1$, thus determining the optimal VM type τ_i for application class i. c_i is the number of containers allocated to class i, whilst m_i and v_i are the container sizes in RAM and vCPUs. On the other hand, M_i and V_i are the total RAM and vCPUs available per Node Manager (i.e., per VM) and h_i is the required concurrency level. With the given conditions the following mathematical programming formulation is proposed:

$$\min_{\mathbf{x},\boldsymbol{\nu},\mathbf{r},\mathbf{d},\mathbf{s}} \quad \sum_{i\in\mathcal{A}} \left(\sigma_{\tau_i} s_i + \delta_{\tau_i} d_i + \rho_{\tau_i} r_i\right) \tag{P1a}$$

subject to:

$$\sum_{j\in\mathcal{V}} x_{ij} = 1, \quad \forall i \in \mathcal{A}, \tag{P1b}$$

$$\mathcal{P}_{i,\tau_i} = \sum_{j\in\mathcal{V}} \mathcal{P}_{ij} x_{ij}, \quad \forall i \in \mathcal{A}, \tag{P1c}$$

$$\delta_{\tau_i} = \sum_{j\in\mathcal{V}} \delta_j x_{ij}, \quad \forall i \in \mathcal{A}, \tag{P1d}$$

$$\sigma_{\tau_i} = \sum_{j\in\mathcal{V}} \sigma_j x_{ij}, \quad \forall i \in \mathcal{A}, \tag{P1e}$$

$$\rho_{\tau_i} = \sum_{j\in\mathcal{V}} \rho_j x_{ij}, \quad \forall i \in \mathcal{A}, \tag{P1f}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in \mathcal{A}, \forall j \in \mathcal{V}. \tag{P1g}$$

$$(\boldsymbol{c},\mathbf{r},\mathbf{d},\mathbf{s}) \in \arg\min \sum_{i\in\mathcal{A}} \left(\delta_{\tau_i} d_i + \sigma_{\tau_i} s_i + \rho_{\tau_i} r_i\right) \tag{P1h}$$

subject to:

$$r_i \le R_{i,\tau_i}, \quad \forall i \in \mathcal{A}, \tag{P1i}$$

$$s_i \le \frac{\eta_i}{1-\eta_i}\left(r_i + d_i\right), \quad \forall i \in \mathcal{A}, \tag{P1j}$$

$$\nu_i = r_i + d_i + s_i, \quad \forall i \in \mathcal{A}, \tag{P1k}$$

$$m_i c_i \le M_i \nu_i, \quad \forall i \in \mathcal{A}, \tag{P1l}$$

$$v_i c_i \le V_i \nu_i, \quad \forall i \in \mathcal{A}, \tag{P1m}$$

$$\mathcal{T}\left(\mathcal{P}_{i,\tau_i}, \nu_i, h_i; Z_i\right) \le D_i, \quad \forall i \in \mathcal{A}, \tag{P1n}$$

$$\nu_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P1o}$$

Copyright © 2017, DICE consortium — All rights reserved

$$c_i \in \mathbb{N}, \quad \forall i \in \mathcal{A} \tag{P1p}$$
$$r_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P1q}$$
$$d_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P1r}$$
$$s_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}. \tag{P1s}$$

Problem (P1) is a bi-level resource allocation problem where the outer objective function (P1a) considers running costs. The first set of constraints, (P1b), associates each class i with only one VM type j, hence the following constraints, ranging from (P1c) to (P1f), pick the values for the inner problem parameters.

The inner objective function (P1h) has the same expression as (P1a), but in this case the prices σ_{τ_i}, δ_{τ_i} and ρ_{τ_i} are fixed, as they have been chosen at the upper level. Constraints (P1i) bound the number of reserved VMs that can be concurrently switched on according to the contracts in place with the IaaS provider. The following constraints, (P1k) add all the VMs available for class i, irrespective of the pricing model. Further, constraints (P1n) mandate to respect the deadlines D_i, as stated in the Service Level Agreement contracts. In the end, all the remaining decision variables are taken from the natural numbers set, according to their interpretation.

The given formulation is particularly difficult to tackle, as it is a bilevel MINLP problem, possibly nonconvex, depending on $\mathcal{T}$. According to the literature about complexity theory [30], integer programming problems belong to the NP-hard class, hence the same applies to (P1). Moreover extending (P1) to the private cloud case would only worsen the situation.

In the following, with some assumptions, the given formulation is going to be relaxed and simplified. Since there is no constraint linking variables belonging to different application classes, it is possible to split this general formulation into several smaller and independent problems, one per class i.

$$\min_{c_i, r_i, d_i, s_i} \quad \rho_{\tau_i} r_i + \delta_{\tau_i} d_i + \sigma_{\tau_i} s_i \tag{P2a}$$

subject to:

$$r_i \leq R_{i,\tau_i}, \quad \forall i \in \mathcal{A}, \tag{P2b}$$
$$s_i \leq \frac{\eta_i}{1-\eta_i}\left(r_i + d_i\right), \quad \forall i \in \mathcal{A}, \tag{P2c}$$
$$m_i c_i \leq M_i\left(r_i + d_i + s_i\right), \quad \forall i \in \mathcal{A}, \tag{P2d}$$
$$v_i c_i \leq V_i\left(r_i + d_i + s_i\right), \quad \forall i \in \mathcal{A}, \tag{P2e}$$
$$\chi_i^h h_i + \chi_i^c \frac{1}{c_i} + \chi_i^0 \leq D_i, \quad \forall i \in \mathcal{A}, \tag{P2f}$$
$$c_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P2g}$$
$$r_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P2h}$$
$$d_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P2i}$$
$$s_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}. \tag{P2j}$$

The objective (P2a) is the total hourly cost for the optimal cluster. Constraint (P2b) bounds the number of reserved VMs that can be turned on, according to the contracts with the IaaS provider. (P2c) limits the number of spot VMs to less than a fraction η of the total machines. Constraints (P2d)-(P2e) ensure that the overall number of containers is consistent with the configured Node Manager capacity. Exploiting the SVR method described in Section D.3 has been possible to rewrite the constraint (P1n) into (P2f), enforcing that the completion time meets an arranged deadline. In the end, (P2g)–(P2j) provide all the variable definition domains.

Copyright © 2017, DICE consortium — All rights reserved

Problem (P2) can be reasonably relaxed to a continuous formulation. This change makes it possible to apply the Karush-Kuhn-Tucker conditions, which are sufficient for optimality due to the problem regularity. Problem (P2) can be additionally simplified by a couple of simple algebraic transformations, thus obtaining the relaxed solution in closed form.

Constraints (P2d) and (P2e) share the same basic structure and are alternative, hence in every feasible solution at most one can be active. Building upon this consideration, it is possible to reformulate them as a single constraint, based on which is the most stringent:

$$c_i \leq \alpha_i \left(r_i + d_i + s_i\right), \text{ where } \alpha_i \triangleq \min\left\{\frac{M_i}{m_i}, \frac{V_i}{v_i}\right\} \tag{8}$$

Moreover, given the price of reserved ρ_i, demand δ_i and spot σ_i VMs for the chosen VM type τ_i and the total number of VMs, of the selected type τ_i, needed to support the required workload, it is trivial to determine the optimal instance mix. Since $\sigma_i < \delta_i$ and $\sigma_i < \rho_i$, the spot VMs are selected first, but respecting the constraint (P2c), then on-demand or reserved are chosen.

According to this and considering only variable ν_i instead of all the VM-related ones, the formulation is:

$$\min_{c_i, \nu_i} \nu_i \tag{P3a}$$

subject to:

$$c_i \leq \alpha_i \nu_i, \qquad \forall i \in \mathcal{A}, \tag{P3b}$$

$$\chi_i^h h_i + \chi_i^c \frac{1}{c_i} + \chi_i^0 \leq D_i, \qquad \forall i \in \mathcal{A}, \tag{P3c}$$

$$c_i \geq 0, \qquad \forall i \in \mathcal{A}, \tag{P3d}$$

$$\nu_i \geq 0, \qquad \forall i \in \mathcal{A}. \tag{P3e}$$

Thus, it is possible to apply the KKT conditions:

$$\begin{aligned} \mathcal{L}\left(c_i, \nu_i\right) = \nu_i &+ \lambda_\alpha \left(c_i - \alpha_i \nu_i\right) + \\ &+ \lambda_\chi \left(\chi_i^h h_i + \chi_i^c \frac{1}{c_i} + \chi_i^0 - D_i\right) + \\ &- \lambda_c c_i - \lambda_\nu \nu_i. \end{aligned} \tag{9}$$

$$\frac{\partial \mathcal{L}}{\partial \nu_i} = 1 - \alpha_i \lambda_\alpha - \lambda_\nu = 0, \tag{10a}$$

$$\frac{\partial \mathcal{L}}{\partial c_i} = \lambda_\alpha - \lambda_\chi \chi_i^c \frac{1}{c_i^2} - \lambda_c = 0. \tag{10b}$$

$$\lambda_\alpha \left(c_i - \alpha_i \nu_i\right) = 0, \quad \lambda_\alpha \geq 0, \tag{11a}$$

$$\lambda_\chi \left(\chi_i^h h_i + \chi_i^c \frac{1}{c_i} + \chi_i^0 - D_i\right) = 0, \quad \lambda_\chi \geq 0, \tag{11b}$$

$$\lambda_c c_i = 0, \quad \lambda_c \geq 0, \tag{11c}$$

$$\lambda_\nu \nu_i = 0, \quad \lambda_\nu \geq 0. \tag{11d}$$

Constraint (P3c) requires $c_i > 0$ and, thanks to (P3b), it also holds $\nu_i > 0$. Thus, $\lambda_c = 0$ and $\lambda_\nu = 0$. Now, equations (10a)(10b) and (10b) can be applied to obtain $\lambda_\alpha > 0$ and $\lambda_\chi > 0$. Hence constraints (P3c) and (P3d) are active in every optimal solution, so:

$$c_i = \frac{\chi_i^c}{D_i - \chi_i^h h_i - \chi_i^0}, \tag{12a}$$

Copyright © 2017, DICE consortium — All rights reserved

Table 7: Additional model parameters for the private cloud

Parameter	Definition
$\mathcal{S}$	Set of homogenous physical servers
M_i	Total memory available per Node Manager associated to applications of class i [GB]
V_i	Total number of vCPUs available per Node Manager associated to applications of class i
$\tilde{M}_{il}$	RAM required by each VM associated to class i at concurrency level l [GB]
$\tilde{V}_{il}$	Number of vCPUs required by each VM associated to class i at concurrency level l
N	Number of homogenous physical server available in the Private Cloud cluster
V	Number of vCPUs of each cluster server
M	RAM of each cluster server [GB]
E	Cost of each physical server of the cluster
p_i	Pecuniary penalty associated to the rejection of one user of class i
C_{il}	Sum of running costs and penalties due to job rejections for class i run at concurrency level l
P_{il}	Penalties due to job rejection associated for each concurrency level l of each class i

Table 8: Additional decision variables for the private cloud

Variable	Definition
w_{il}	Set of logical variables used to choose one concurrency level per class
y_k	Binary variable equals to 1 when the physical machine k is turned on
n_{ilk}	Logical variable used to determine the number of VMs allocated in node k for class i at concurrency level l

$$\nu_i = \frac{c_i}{\alpha_i} = \frac{1}{\alpha_i} \frac{\chi_i^c}{D_i - \chi_i^h h_i - \chi_i^0}. \tag{12b}$$

Those two equations, together with the SVR technique described in Section D.3, are exploited in the first phase of the HC to calculate the initial solution.

D.5 Private Cloud Extension

Above a general formulation of the resource provisioning problem has been exposed. Here is shown how it can cover also the private case. When an in house cluster is considered, the separation among different pricing models disappears (i.e. reserved, on-demand, spot distinction), but other facility costs could be taken into account. Moreover, in order to contemplate the possibility to exhaust the cluster computational capacity, these costs comprehend also penalties bound to job rejections. This is the AC case described in Section 3.1 with the assumption introduced in Section D.2. Furthermore, AC enlarges the feasible region of the resource provisioning problem by lowering the concurrency level. In this section two different models, supporting the private cloud with AC, are presented. Both models aim to find h_i, that is the optimal concurrency level for each class, respecting the application QoS constraints. As stated in Section D.2, h_i is subject to the restrictions specified by the system architect: $H_i^{low} \leq h_i \leq H_i^{up}$. Both models minimize the deployment cost, but, the one presented in Section D.5.1 is simpler and does not consider how to distribute the required VMs among the physical nodes, in contrast, the model described in Section D.5.2 takes into account also this fact, generating a more accurate solution.

The additional model parameters used to cover the private cloud case are introduced in Table 7, whilst Table 8 summarizes the decision variables for this case.

D.5.1 Multi-dimensional Knapsack Problem

Every application class i has an associated set of possible concurrency levels $\mathcal{H}_i$. The penalty associated to a class i with a given concurrency level l, remembering that $h_i = l$ can span from H_i^{low} to H_i^{up}, can

Copyright © 2017, DICE consortium — All rights reserved

be calculated as $\left(H_i^{up} - l\right) \cdot p_i$, where p_i is the hourly cost of rejecting one user of class i. C_{il} are the sum of running costs and penalties due to job rejections for class i run at concurrency level $l \in \mathcal{H}_i$. w_{il} are logical variables used to choose exactly one concurrency level per class. Moreover, $\tilde{M}_{il}$ and $\tilde{V}_{il}$ are the total RAM and vCPUs required by each VM associated to class i at concurrency level l. Notice that different VM types $v \in \mathcal{V}$ can be associated to each couple (i, l). In the end, N is the number of physical nodes available in the cluster, while M and V are the node capacity in RAM and CPU.

$$\min_{\mathbf{w}} \quad \sum_{i \in \mathcal{A}} \sum_{l \in \mathcal{H}_i} C_{il} w_{il} \tag{P4a}$$

subject to:

$$\sum_{l \in \mathcal{H}_i} w_{il} = 1, \quad \forall i \in \mathcal{A}, \tag{P4b}$$

$$\sum_{i \in \mathcal{A}} \sum_{l \in \mathcal{H}_i} \tilde{M}_{il} \nu_{il} w_{il} \leq MN, \tag{P4c}$$

$$\sum_{i \in \mathcal{A}} \sum_{l \in \mathcal{H}_i} \tilde{V}_{il} \nu_{il} w_{il} \leq VN, \tag{P4d}$$

$$w_{il} \in \{0, 1\}, \quad \forall i \in \mathcal{A}, \forall l \in \mathcal{H}_i. \tag{P4e}$$

The objective (P4a) is to minimize the total of running costs and penalties. (P4b) ensure that exactly one concurrency level per class is chosen. Constraints (P4c) and (P4d) enforce that the cluster capacity fits the required VMs, in terms of total cores and total memory. Finally, (P4e) give the definition domain for all the w_{il} variables.

D.5.2 Bin Packing Problem

The previously described model considers only the total of resources, cores and memory, available in the cluster. Bin packing, on the other hand, takes also into account how the VMs of each class should be distributed among the physical nodes. For example, a VM with four cores cannot be split in two different physical machines, having two cores in one server and two cores in another.

Also in this model the cluster is composed of N homogeneous physical servers collected in the set $\mathcal{S}$. Introducing n_{ilk} as the number of VMs allocated on node $k \in \mathcal{S}$ for class i at concurrency level l, allows to consider how the VMs have to be spread along the cluster. The aggregate penalty associated to class i and concurrency level l is P_{il}. In the end, y_k is a logical variable that is equal to 1 when the physical machine k is turned on and E is the cost to run one physical machine. Let $\beta \triangleq \max \left\{ \frac{M}{\tilde{M}_{il}}, \frac{V}{\tilde{V}_{il}} \right\}_{i \in \mathcal{A}, l \in \mathcal{H}_i}$ and $\nu_{il} = r_{il} + d_{il} + s_{il}$. The problem can be formulated as:

$$\min_{\mathbf{n},\mathbf{w},\mathbf{y}} \quad \sum_{i \in \mathcal{A}} \sum_{l \in \mathcal{H}_i} P_{il} w_{il} + \sum_{k \in \mathcal{S}} E y_k \tag{P5a}$$

subject to:

$$\sum_{l \in \mathcal{H}_i} w_{il} = 1, \quad \forall i \in \mathcal{A}, \tag{P5b}$$

$$\sum_{i \in \mathcal{A}} \sum_{l \in \mathcal{H}_i} n_{ilk} \leq \beta y_k, \quad \forall k \in \mathcal{S}, \tag{P5c}$$

$$\sum_{k \in \mathcal{S}} n_{ilk} = \nu_{il} w_{il}, \quad \forall i \in \mathcal{A}, \forall l \in \mathcal{H}_i, \tag{P5d}$$

$$\sum_{i \in \mathcal{A}} \sum_{l \in \mathcal{H}_i} \tilde{M}_{il} n_{ilk} \leq M, \quad \forall k \in \mathcal{S}, \tag{P5e}$$

$$\sum_{i \in \mathcal{A}} \sum_{l \in \mathcal{H}_i} \tilde{V}_{il} n_{ilk} \leq V, \quad \forall k \in \mathcal{S}, \tag{P5f}$$

$$w_{il} \in \{0, 1\}, \quad \forall i \in \mathcal{A}, \forall l \in \mathcal{H}_i, \tag{P5g}$$

Copyright © 2017, DICE consortium — All rights reserved

$$y_k \in \{0,1\}, \quad \forall k \in \mathcal{S}, \tag{P5h}$$
$$n_{ilk} \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \forall l \in \mathcal{H}_i, \forall k \in \mathcal{S}. \tag{P5i}$$

The objective function (P5a) sums the penalties associated to job rejections and the energy costs to sustain the physical computational nodes. Constraints (P5b) enforce that every class has a single concurrency level. (P5c) set y_k to 1 if any VMs are allocated on node k, whilst (P5d) ensure that, overall, exactly ν_{il} VMs are running. Further, constraints (P5e) and (P5f) bound the memory and CPU requirements of VMs allocated onto the same physical node. At last, (P5g)–(P5i) are the variable definition domains.

D.6 Scenarios Fitted by the Algorithms

The algorithms discussed in Section 4.2 allow to solve the capacity planning problem for Big Data applications in several different situations.

Some interesting use cases, which covers very different scenarios, can be derived by changing the parameters of the introduced algorithms as described in the next paragraphs. Setting $\eta_i = 0$ prevents from using spot VMs, besides $R_i = 0$ impedes the usage of reserved VMs. In all the public clouds cases is assumed that penalties due to job rejections are more expensive than deploying the required machines, hence in such scenarios the AC mechanism is disabled by setting $h_i = H_i^{up}$ in problem (P2). Notice that, in all the private clouds cases, $R_i = \eta_i = 0$ and $\rho_i = \sigma_i = 0$, since companies running on-premises clusters are only interested in minimizing operational costs associated to physical nodes.

In particular our approach allows to determine the optimal deployment solution in the following scenarios:

In public clouds:

- When the system ARCHITECT has already paid for some reserved machines, which, however, are not sufficient to sustain peaks in the workload. In such a case, $R_i = \bar{R}_i > 0$ and the interest is to determine the number of additional on-demand and, possibly, spot VMs needed to absorb the temporary increase in load. This can be achieved by Algorithm 1 with $h_i = H_i^{up}$
- When the system ARCHITECT has not already leased any cloud computing resources and want to know which and how many VMs are sufficient to hold the load. This case is covered setting $R_i = \eta_i = 0$ and $\rho_i = \sigma_i = 0$, then solving for the minimum number of machines with Algorithm 1.

In private clouds:

- When the system ARCHITECT has already a pool of physical computing resources and wants to evaluate a given the performance of given DIA over it. In this case AC is enabled and $H_i^{low} < H_i^{up}$, thus possibly leading to job rejections if the cluster capacity is not enough to accommodate all the required workload, obviously with $p_i > 0$. Thus, (P4) or (P5) are exploited to select the optimal concurrency level for each class and the solution is obtained through Algorithm 2.
- When the in-house system has not already be provisioned, and the ARCHITECT want to know the type and how many VMs are sufficient to sustain the given workload. Since job rejection is not taken into account, this case can be fulfilled by exploiting Algorithm 1 setting $R_i = \eta_i = 0$ and $\rho_i = \sigma_i = 0$.

D.7 Modeling Storm Technology

Storm technology is partially supported by **D-SPACE4Cloud** and it has been introduced to demonstrate that the tool is extensible and new technologies can be added with a limited effort. Storm has been chosen since it is of interest for DICE ATC and PRO case studies and also because it is based on streaming, which has totally different characteristics with respect to the batch execution of MapReduce and Spark applications considered so far.

Copyright © 2017, DICE consortium — All rights reserved

For DIAs based on Storm, **D-SPACE4Cloud** supports only public cloud deployments and allows to specify constraints predicating on the utilization of the running cluster. For what concerns the mathematical programming aspects, Equation 6 is replaced by

$$U_i = \mathcal{U}\left(\mathcal{P}_{i,\tau_i}, \nu_i, \Lambda_i\right), \quad \forall i \in \mathcal{A}, \tag{13}$$

which estimates the cluster utilization U_i as a function of the DIA profile P_{i,τ_i}, the total number of cores of the cluster ν_i and the frequency Λ_i of the set of spouts the DIA is registered (note that Λ_i denotes a vector whose components are the arrival rates of the tuples of the considered spouts).

Similarly, Equation P1n constraints in problem (P1) is replaced by:

$$U_i \leq \overline{U}_i, \quad \forall i \in \mathcal{A}, \tag{14}$$

which guarantees that the cluster utilization is below a threshold $\overline{U}_i$ set a priori.

The utilization U_i is estimated by relying on the simulation models described in DICE Deliverables D3.2, D3.3, and D3.4, which are evaluated through GreatSPN. Finally, Algorithm 1 starts from a cluster including three nodes and increases/decreases the number of nodes by one step. Since, the objective function is convex, if the VM type is fixed Algorithm 1 finds again the globally optimal solution for problem P1. The solution for Storm is only less efficient, since the search cannot exploit a promising initial solution and the speedup of the hyperbolic jump.

Copyright © 2017, DICE consortium — All rights reserved

Listing 5: Example JSON file with ML profile

```
{
  "Q1": {
    "b": -0.0054638,
    "mu_t": 432230.122,
    "sigma_t": 170134.948,
    "mlFeatures": {
      "avgTask_S0": {
        "w": 0.012694,
        "mu": 1632.769,
        "sigma": 159.488
      },
      "avgTask_S1": {
        "w": -0.016927,
        "mu": 1527.671,
        "sigma": 206.061
      },
      "x": {
        "w": 0.71995,
        "mu": 0.042528,
        "sigma": 0.021299
      },
      "h": {
        "w": 0.12854,
        "mu": 2511.447,
        "sigma": 409.724
      }
    }
  }
}
```

E Machine Learning Model Input Files

This section discusses examples of the ML model file, which can be provided with the premium version of **D-SPACE4Cloud** and, usually, allows to speed up significantly the optimization time providing a promising initial solution to the HC algorithm. Such a JSON object is meant to provide all the parameters relevant for the application of a support vector regression (SVR) model, as described in Section D.3.

Listing 5 contains a dictionary that associates queries, here `Q1`, to the respective ML profiles. These, in turn, contain all the parameters needed to apply a linear SVR model: the mean (`mu`) and standard deviation (`sigma`) used for data normalization, for all the features and for the response time to provide as output, in addition to the constant term (`b`) and the coefficients (`w`) of the regression line.

Recall that the goal of SVR is to fit a regression line of the form:

$$t = \mathbf{w}^{\mathrm{T}}\boldsymbol{\zeta} + b,$$

so that most data points lie in a stripe centered around it, minimizing both the line coefficients and the remaining points distance from the stripe. Here t is the response time to predict, whilst $\boldsymbol{\zeta}$ is the vector containing all the features described in the `mlFeatures` dictionary. In this example, both $\mathbf{w}$ and $\boldsymbol{\zeta}$ have four elements.

Moreover, due to numerical analysis considerations, it is advisable to normalize both the features and the predicted variable when using SVR. **D-SPACE4Cloud** obtains normalized features via Z scores:

$$\forall i \quad z_i = \frac{y_i - \mu_i}{\sigma_i},$$

whence the need for means and standard deviations in this JSON.

Since the main purpose of the optimizer is to determine the optimal concurrency and resource allocation, the dictionary `mlFeatures` must at least provide the features `h`, which represents the contribution given by concurrency, and `x`, stating the influence of additional CPU cores on the execution time.

Copyright © 2017, DICE consortium — All rights reserved