Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements

DICE

DICE Optimization Tools — Initial Version

Deliverable 3.8

Deliverable:	D3.8
Title:	Optimization Tools — Initial Version
Editor(s):	Michele Ciavotta (PMI), Danilo Ardagna (PMI).
Contributor(s):	Michele Ciavotta (PMI), Eugenio Gianniti (PMI), Vitor Soares Lopes (IMP).
Reviewers:	Matej Artac (XLAB), Richard Brown (ATC).
Type (R/P/DEC):	DEM
Version:	1.0
Date:	**26-07-2016**
Status:	**Final version**
Dissemination level:	Public
Download page:	`http://www.dice-h2020.eu/deliverables/`
Copyright:	Copyright © 2016, 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

The current deliverable is the first of the two deliverables (D3.8, and D3.9) reporting the status of development activities related to T3.4 that is the realization of DICE Optimization Tools (**D-SPACE4Cloud**). Furthermore, in this document, we present a first reference to install and use the prototype tool to be developed in the framework of this task.

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 deadlines under different levels of cluster congestion. More precisely, **D-SPACE4Cloud** takes 1. a DICE annotated model in form of DDSM, 2. the Service Level Agreement (SLA) to be validated, and 3. a description of the execution environment (list of providers, list of virtual machine (VM) types or a description of the computational power available in house, application expected profiles) and uses them to generate (via DICE transformation tools) a suitable performance model that is then evaluated and optimized. The optimization consists in finding the less expensive cluster configuration able to guarantee the application to be completed before a user defined deadline. The architect can analyze the application behavior under different conditions; she 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. The outcome of the process is a cluster configuration with the selection of the cloud provider (in case the public cloud scenario is considered), VM types and number and costs. Costs are obtained considering cloud providers' own pricing models or electricity costs (in case the private cloud scenario is analyzed).

D-SPACE4Cloud encompasses different modules that communicate each other by means of RESTful interfaces or SSH following the Service Oriented Architecture (SOA) paradigm. In particular, it features a presentation and orchestration service (referred to as *frontend*) and an horizontally scalable optimization service (referred to as *backend*), which makes use of third-party services as RDBMS, simulators and mathematical solvers.

Glossary

DAG	Directed acyclic graph
DICE	Data-Intensive Cloud Applications with iterative quality enhancements
DIA	Data-Intensive Application
DPIM	DICE Platform Independent Model
DTSM	DICE Platform and Technology Specific Model
DDSM	DICE Deployment Specific Model
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
M2M	Model to Model transformation
QA	Quality Assurance
UML	Unified Modelling Language
QN	Queueing network
SPN	Stochastic Petri net
SWN	Stochastic Well-formed net
MINLP	Mixed integer nonlinear programming
QoS	Quality of Service

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

The crux 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 user of the DICE methodology is 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 model that, eventually, undergoes 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, expressed in terms of job[1] deadlines.

This document describes the initial version of **D-SPACE4Cloud**, which is developed in the framework of WP3 Task 3.4 and is published as an open source tool in the repository of Polimi on GitHub.[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 the 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 quality verification and optimization tools.

Task T3.4 focuses on the development of the optimization tool to tackle to the *resource provisioning* problem for DIAs in the specific context of private and public clouds (see Section 4.2 for more details about the problem).

The outcome of tasks T2.1 and T2.2 are DIA topologies, which consist of two parts, defined in the DDSM annotated diagrams: *(i)* the graph-based representation of the application under development and *(ii)* a set of non-functional requirements that complement the application model and allow the designer to perform the optimization under different levels of concurrency.[4] Such diagram represent the ground basis for the research carried out in T3.4.

This deliverable reports on the research activity carried out in task T3.4, whose goal is to develop and implement a set of tools supporting the identification of optimized DIA deployments. Such tools are based on well-known results from the combinatorial optimization field. Nonetheless, given the inherent complexity of the problem faced, the design and implementation of an efficient and effective technique represents a challenge in itself and an advance for the research field. Therefore, in this deliverable some space is dedicated to describing a local search based algorithm, hybridized with exact methods and simulation. As a matter of fact, simulation models (developed within T3.2) play a role of paramount importance in the optimization process as it is able to estimate with a certain accuracy the execution time of the DIA at design time under different levels of concurrency (see deliverable D3.1, *Transformations to*

[1] Here we consider MapReduce jobs, future work will also take into account Spark, where all the computation that takes place between two subsequent *actions* constitutes a job.

[2] `https://github.com/deib-polimi/diceH2020-space4cloudsWS`

[3] `https://github.com/deib-polimi/diceH2020-space4cloud-webGUI`

[4] With concurrency level we mean the number of users accessing the system at the same time.

analysis models). Furthermore, this document provides an architectural and behavioral description of the initial version of the optimization tool, along with the required inputs and the expected outputs. Finally, a validation of the quality of the solution in terms of deadline fulfillment is obtained through performance models, namely, Stocastic well-formed networks (SWNs) and Queueing networks (QNs).

The contributions of Task T3.4 are:

1. the definition of the resource provisioning problem in terms of shared Cloud cluster for concurrent DIAs

2. the evaluation of the quality of the solution through QNs and SWNs

3. the design and implementation of state-of-the-art algorithm enabling the user to reason about different cluster configuration and DIA topologies.

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 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 maximum execution time (hereinafter called *deadline*) 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, 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 model, launch the optimization algorithm, and taking track of 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 the requirements related to the Optimization Tools. Section 3 introduced **D-SPACE4Cloud** and shows its interaction with the DICE framework. Section 4 provides some context about DIAs, it introduces the resource provisioning problem and presents the underlying performance models used to simulate DIA behavior under different levels of concurrency. Section 5 discusses the implementation of **D-SPACE4Cloud**, with particular focus on the backend service. Section 6 shows some practical experiments to validate the approach and discusses future achievements. Finally, Appendix A provides the installation and usage manual, Appendix B provides further details on the mathematical formulation of the optimization problem, whereas Appendix C provides documentation specifically addressed to developers.

2 Requirements and usage scenarios

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 current **D-SPACE4Cloud** prototype.

2.1 Tools and actors

As specified in 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 optimization, and employs the following tools:

- Transformation Tools
- Simulation Tools
- Verification Tools
- Optimization Tools **D-SPACE4Cloud**, which takes as input some partially specified DDSM models produced by the application designers, and determines the minimum cost configuration such that the SLA associated with the DIA is met.

In the rest of this document, we focus on the tools related to Task T3.4, i.e., **D-SPACE4Cloud**. According to 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.

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 can be summarized as follows [2, p. 104]:

ID:	UC3.3
Title:	Optimization of the deployment from a DDSM DICE annotated UML model with reliability and performance constraints
Priority:	REQUIRED
Actors:	ARCHITECT OPTIMIZATION_TOOLS, SIMULATION_TOOLS
Pre-conditions:	There exists a DDSM level UML annotated model (where the number and possibly type of VMs are not specified). Cost are stored in the OPTIMIZATION_TOOLS internal resource DB
Post-conditions:	The ARCHITECT starts from a partial DDSM model and reasons about the optimal resource allocation considering the trade off cost/requirements. Multiple technology are analyzed by providing multiple DDSMs throught what-if scenarios.

The requirements listed in [1] are the following:

ID:	R3.8
Title:	Cost/quality balance
Priority of accomplishment:	Must have
Description:	The OPTIMIZATION_TOOLS will minimize deployment costs trying to fulfill reliability and performance metrics (e.g., map-reduce jobs execution deadlines).

ID:	R3.10
Title:	SLA specification and compliance
Priority of accomplishment:	Must have
Description:	OPTIMIZATION_TOOLS MUST permit users to check their outputs against SLA's included in UML model annotations. If an SLA is violated the tools will inform the user.

ID:	R3.11
Title:	Optimization timeout
Priority of accomplishment:	Could have
Description:	The OPTIMIZATION_TOOLS MUST explore the design space and should accept the specification of a timeout and return results gracefully when this timeout is expired

3 Optimization tool overview

D-SPACE4Cloud (DICE Optimization Tool) is the optimization tool integrated in the DICE IDE. It consists of two main components: frontend and a backend. Details on the overall architecture and on the internals of each component can be found in Section 5. Suffice to say that the frontend exposes a graphical interface designed to facilitate the interaction with the end user while the backend implements a strategy aimed at optimizing semi-defined DTSM diagrams.

DDSM is only semi-definite as the ARCHITECT has still to decide VMs types and numbers and evaluate performance and costs. **D-SPACE4Cloud** is the tool appointed to help the user in such an important task. In the current version of the tool the DDSM model[5] is not the only input to be provided to the optimization tool; it goes along with a description of the execution environment (list of providers, list of VM types or a description of the computational power available in house, application expected profiles). The reader is referred to Section C.4 for a complete description of the input files and a example of their format.

In the final version of the tool, the model-to-model (M2M) transformation mechanism implemented within the DICE IDE will be exploited to generate a suitable performance model (SWN or QN) to be used to predict the expected execution time of the DIA; at the time of writing the performance is evaluated via a set of models embedded in the tool. Such modes, 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 initial 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 Section 5.3 for a detailed comparison of the alternatives.

The set of third-party tool needed to execute **D-SPACE4Cloud** includes also AMPL [5], a tool for modeling and handling optimization problems in form of mathematical models. In a nutshell, AMPL provides 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. Future releases of the optimization tool will rely on full open source optimization modelling tools and solvers, such as CMPL [6] or GLPK [7].

The sequence diagram depicted in Figure 1 describes the interactions occurring among the component of DICE optimization tool ecosystem. Through the IDE and the graphical interface provided by **D-SPACE4Cloud** the ARCHITECT uploads the DICE DDSM model along with the other models describing the scenario under study. Several experiments can be submitted; the frontend component manages the runs, carrying out them in parallel where 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 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 deadlines associated with jobs. This phase is designed to spawn swiftly an initial solution that could hopefully set the successive local search phase in a promising position within the space of possible solution (also referred to as *Solution Space*). 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 job. This task is executed in parallel where possible, considering job deadlines 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.

[5]Note that at the time of writing DDSMs are still under definition.

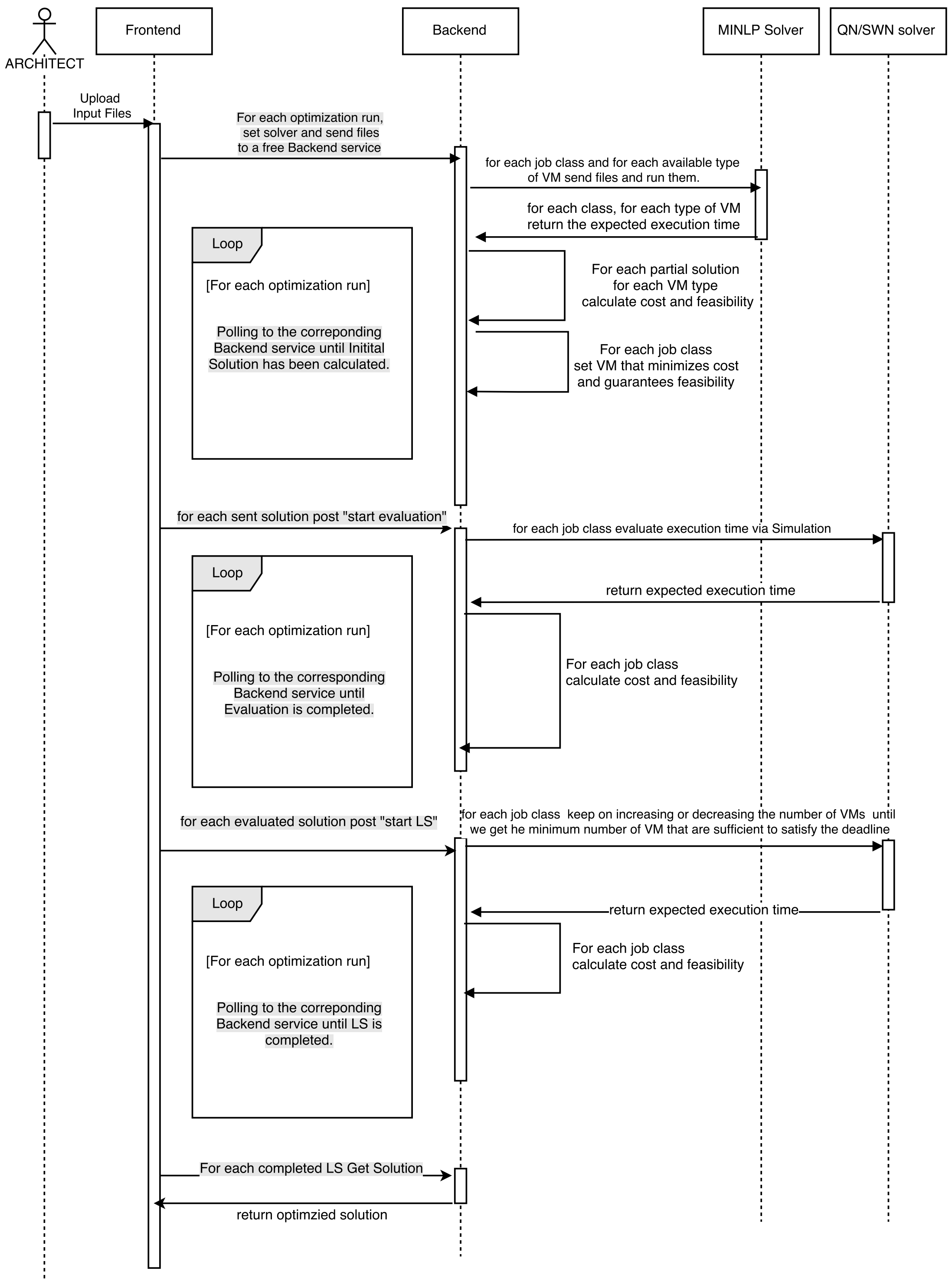

ARCHITECT
Frontend
Backend
MINLP Solver
QN/SWN solver
Upload Input Files
For each optimization run, set solver and send files to a free Backend service
for each job class and for each available type of VM send files and run them.
for each class, for each type of VM return the expected execution time
For each partial solution for each VM type calculate cost and feasibility
For each job class set VM that minimizes cost and guarantees feasibility
Loop
[For each optimization run]
Polling to the correponding Backend service until Initital Solution has been calculated.
for each sent solution post "start evaluation"
for each job class evaluate execution time via Simulation
return expected execution time
For each job class calculate cost and feasibility
Loop
[For each optimization run]
Polling to the corresponding Backend service until Evaluation is completed.
for each evaluated solution post "start LS"
for each job class keep on increasing or decreasing the number of VMs until we get he minimum number of VM that are sufficient to satisfy the deadline
return expected execution time
For each job class calculate cost and feasibility
Loop
[For each optimization run]
Polling to the correponding Backend service until LS is completed.
For each completed LS Get Solution
return optimzied solution

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

3.1 Dependencies on other components of DICE IDE

As seen, DICE Optimization tools heavily rely on transformation and simulation tools to achieve the goal (optimization of DDSM diagrams). DICE Transformation tools, as a matter of fact, generate from the UML DICE-profiled model, via suitable M2M transformations, performance models; DICE Simulation tools, instead, are in charge of validating such models and orchestrating external tools in order to evaluate them in terms of performance and reliability.

It is important to reiterate that, while the final version of the optimization tools will use the APIs exposed by the transformation and simulation tools to generate an initial performance model, which will be modified during the local search phase, and evaluate new candidate solutions, the version presented in this deliverable employs its own (fixed) performance models and implements its own connectors to communicate with external model solvers (for more details see Section 5.2).

3.2 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 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 (see Section 4) 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 might also like to know which cloud provider is cheaper to choose, considering that providers have also different pricing models. For this reason, she has to feed the tool with the DDSM diagram but also with a list of providers, and a list of VM types for each provider, and a *profile* for each machine. A profile is a set of information able to describe the performance behavior of the application for a certain virtualized resource. Typical pieces of information in the profile are: number of maps, number of reducers, average map time, average reduce time, etc. For more information refer to Section 4.2
- **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). 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. In is worth to notice, however, that in case of job rejection it is always possible to generate a feasible solution for the problem.
  - **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.

Another aspect not to be underestimated is that data-centers are made of physical machines each with a certain amount of memory and cores; considering the sum of the memory and CPUs as the capacity of the data-center is an approximation that in some cases can lead to generating theoretically feasible provisioning that are actually infeasible. DICE Optimization tool considers also this case and the ARCHITECT can provide a description of the physical machines to generate a more precise solution (this feature will be actually introduced in the next release). Finally, for in house cluster it can be difficult to identify the right cost function, as it can depend on many factors. In the development agenda of the optimization tool a simplified cost model including electricity costs and eventually acquisition costs is considered. Note that the initial version of the **D-SPACE4Cloud** implements only the public cloud deployment scenario.

4 Optimizing the Deployment of Data-intensive applications

This section introduces the reader to the problem of optimizing the deployment of DIAs at design time. In particular, Section 4.1 presents Hadoop 2.x as the reference Big Data ecosystem for the problem; useful concepts to understand the problem are given there. Section 4.2 represents the core of this Section; it presents in detail the resource provisioning optimization problem for the deployment of Big Data cluster in private or public clouds. Finally, Section 4.3 addresses the issue of estimating the completion time of a DIA and QN models are presented.

4.1 Reference technology models

The current version of **D-SPACE4Cloud** supports the optimization of Hadoop deployments. Spark and Storm will be supported by the final release of the optimization tool. The Hadoop framework [8] allows for parallel and distributed computing on large scale clusters of commodity hardware, focusing on batch processing of huge datasets. Furthermore, it guarantees beneficial properties of fault-tolerance, scalability, and automatic parallelization and distribution of computation across the cluster, at the expense of a simple yet rigid programming paradigm. MapReduce jobs are composed of two main phases, namely, Map and Reduce. The former takes as input unstructured data from the Hadoop Distributed File System (HDFS), filtering them and performing a preliminary elaboration, according to the instructions in a user-defined function. The intermediate results are returned as key-value pairs, grouped by key in the Shuffle phase and distributed across the network, so as to provide each node taking part in the Reduce phase with all the values associated with a set of keys. In the end, every reduce node applies a second user-defined function to complete data elaboration and outputs to HDFS.

In more recent versions, Hadoop 2.x allocates resources relying on a distributed resource management framework: Yet Another Resource Negotiator (YARN). This module features different entities to take care of resources at various scales. A first class of components, the Node Managers, handles resources on each computational node, partitioning them as containers. The central Resource Manager, instead, has the role of providing resources for the execution of jobs, based on a configurable scheduling policy. However, all the duties dealing with specific jobs are delegated to Application Masters. When a job is submitted for execution, the Resource Manager bootstraps a container for the corresponding Application Master, then the latter requests a number of containers to perform the computation. According to the current state of the cluster and the scheduling policy, the Resource Manager allows the Application Master to exploit containers.

It is possible to supply a custom scheduler for the Resource Manager to use; nonetheless the Hadoop project provides three ready general purpose alternatives: the *FIFO*, *Fair*, and *Capacity schedulers*. The FIFO scheduler is very basic and follows the scheduling policy adopted by Hadoop 1, with a single global queue served on a first-in/first-out basis. This could lead to job starvation and prevent from sharing clusters, since it does not allow to enforce quotas for different users. The Fair scheduler addresses these shortcomings by guaranteeing a fair share of resources to every user so as to prevent job starvation and allowing to create several queues, in order to easily control resource allocation at different granularity levels. Eventually, the Capacity scheduler boasts features similar to those provided by the Fair one, except that leaf queues are managed on a FIFO basis rather than fair share. In the following, we will focus on modeling clusters adopting the Capacity scheduler.

4.2 Modeling assumptions

In this section we aim at introducing a general overview of the addressed optimization problem. We envision the following scenario, wherein a company needs to set up a cluster to carry out efficiently a set of DIAs. A Hadoop 2.x cluster featuring the YARN Capacity Scheduler and running on a public cloud Infrastructure as a Service (IaaS) is considered a fitting technological solution for the requirements of the company. Different classes gather applications that show a similar behavior. The goal is to meet Quality of Service (QoS) requirements, specifically a prearranged concurrency level (i.e., the number of concurrent users) bound to complete within a deadline. In order to obtain this objective, it is possible to act upon the cluster composition and size, in terms of type and number of VMs.

Moreover, YARN is configured in a way that all the available cores can be dynamically assigned to either Map or Reduce tasks. Finally, in order to limit the risk of data corruption and according to the practices suggested by major cloud vendors [9, 10], the datasets reside on an external storage infrastructure [11, 12] accessible at quasi-constant time.

As, in general, IaaS providers feature a limited, but possibly large, catalog of VM configurations that differ in features and cost, making the right design-time decision poses a challenge that can lead to important savings throughout the cluster life-cycle. In this scenario, we consider a pricing model derived from *Amazon EC2* [13]. The provider offers: 1) *reserved* VMs, for which it adopts a one-time payment policy that grants access to a certain number of them for the contract duration, either at a strongly discounted hourly fee or without further variable costs; and 2) *on-demand* VMs, which customers can buy at a higher price to compensate peaks in the incoming workload, without long term commitment. In order to obtain the most cost-effective configuration, we rely on reserved VMs for the bulk of computational needs and complement them with on-demand instances. Once determined the VM type and the required number of instances, it is easy to compute the overall hourly cluster cost of the configuration.

Reducing the operating costs of the cluster by using efficiently the leased virtual resources is in the interest of the company. This translates into a Resource Provisioning problem where the renting out costs must be minimized subject to the fulfillment of QoS requirements, namely, observing per-class concurrency levels and meeting prearranged deadlines. In the following we assume that the system can be represented as a closed model [14], i.e., that users work interactively with the system and run another job after an exponentially distributed think time.

In order to rigorously 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 approach presented in [15] it is possible to derive from the Hadoop logs a *job profile*, that is a concise behavior characterization for each class. In particular, every job class is characterized by the overall number of Map and Reduce tasks, plus several parameters that quantify task durations and depend on the chosen VM type as well. For a more detailed overview of the considered parameters (which are already included in the DICE profile, see deliverable D2.1 *Design and quality abstractions - Initial version*) the reader is referred to Appendix B.

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. Our configuration 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, where 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, we can say that the ultimate goal of the proposed approach is to determine, for all the classes, the optimal VM type, number, and pricing model, such that the sum of costs is minimized, while the deadlines and concurrency levels are met.

4.3 Performance models

Within **D-SPACE4Cloud**, we make use of both a SWN and a QN model to estimate average MapReduce job completion times, assuming that the YARN Capacity Scheduler is used. In the following, we do not make any assumptions on the configuration of the Capacity Scheduler, according to the considerations presented in Section 4.2. The 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, our model provides the mean completion time for every job class. On the other hand, if the scheduler is configured in work conserving mode, then the completion time we obtain is an approximation due to possible performance gains when resources are exploited by other classes instead of lying idle. Exploiting the node labeling mechanism, we enforce that jobs run on their optimal VM type in any case, so as to avoid cases where some tasks running on improper nodes become bottlenecks and disrupt both the lender and borrower class performance.

SWNs are described in DICE Deliverable D3.1. Details about the QN model can be found in Appendix D.

5 DICE Optimization Tool Architecture

D-SPACE4Cloud follows the Service Oriented Architecture (SOA) pattern. Its architecture, 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 a HTML/CSS based GUI. Further, the frontend is in charge of managing several concurrent optimization runs keeping track of the launched experiments. The fronted is already integrated within the DICE IDE. More details about the frontend are provided in Section 5.1.

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 Section 4.2. 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. The backend component is described in Section 5.2.

Finally, the third tier (described in Section 5.3) encompasses a set of third-party utilities providing different services. 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 [16]. Other services in this layer are a mathematical non linear solver, a SWN or QN simulator managed via SSH connection.

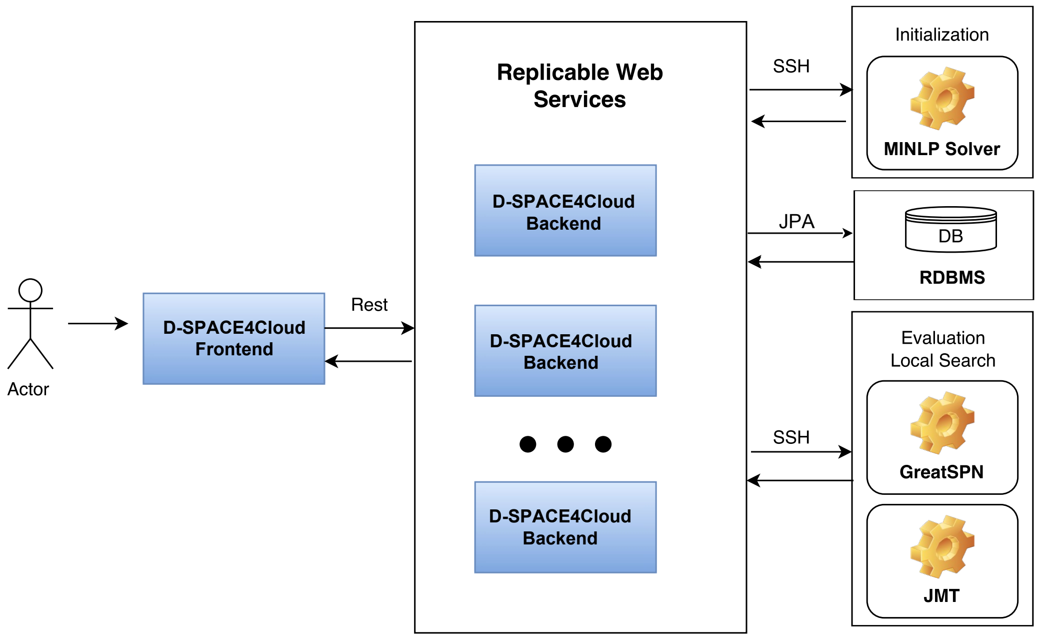


Figure 2: **D-SPACE4Cloud** three-tier architecture

5.1 Frontend service

The fist task carried out by the frontend service 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. Screenshots of **D-SPACE4Cloud** web interface are proposed in Figures 3, 4 and 5. In particular, Figure 3 is the principal interface with the user; we can see four square regions:

- upper-left corner—this square leads to a selection menu for setting up a new optimization considering the scenario of private cloud (see Section 3.2, which will be available with the next release).

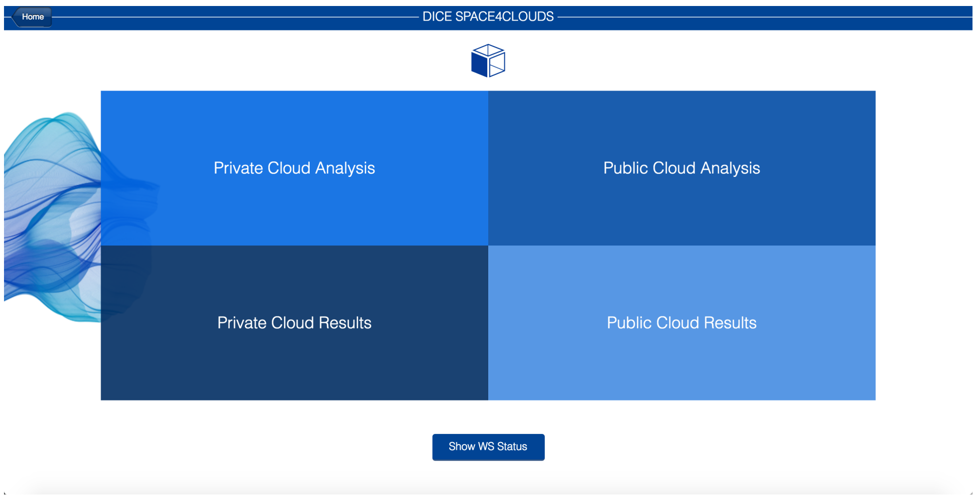


Figure 3: **D-SPACE4Cloud** web GUI: main page

Figure 4 shows some details of the analysis that the tool is preparing to run; in particular, the user can decide to execute the experiment with or without Admission Control. The successive form is presented in Figure 5, where the user can select the appropriate files and upload them to the frontend service.

- lower-left corner—results for the private cloud scenario can be accessed through this square (also this feature will be available with the next release).
- upper-right corner—this square enables the end user to run analyses considering the public cloud scenario.
- lower-right corner—results for the public cloud scenario can be accessed through this square. Figure 6 shows an example of the **D-SPACE4Cloud** page that displays running, aborted and finished experiments, allowing, in the latter case, the user to download them.

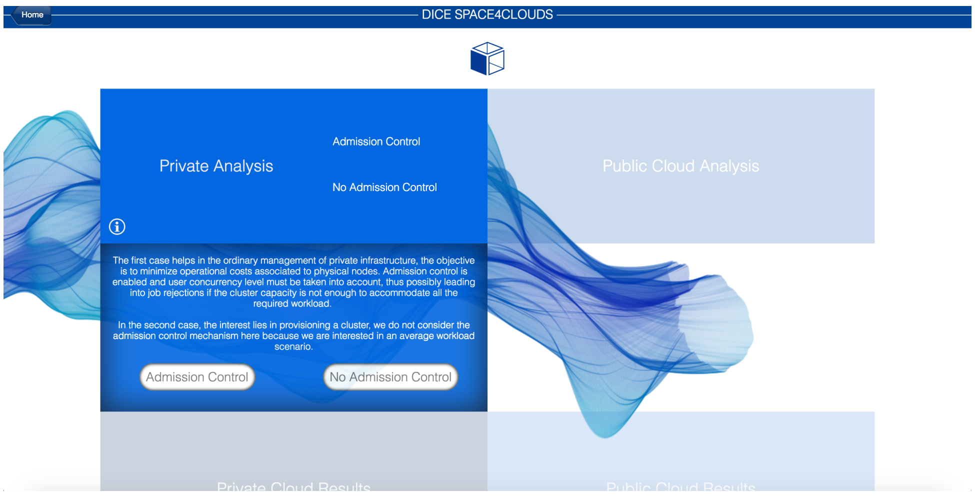
The other duty of the Frontend service is to manage the experiments running them in parallel whether it is possible. In order carry out this task it implements a component called *SimulationsOptManager* (see Figure 7) that handles the list of active Backend services, load-balancing the experiments on them according to a FIFO policy.

It is important to notice that, the Backend service embeds a FSM (presented in Section C.2) that can be controlled via a REST interface. The SimulationsOptManager has to know the details of this machine and manages it to perform the optimization. The interested reader is referred to Section C.3 for the activity diagram of the optimization process, which includes all the REST interactions required to carry out the optimization.

5.2 Backend service

Figure 8 depicts the main elements of the **D-SPACE4Cloud** backend that comes into play in the optimization scenario. The tool is a RESTful web service that takes in input a description of the considered problem, consisting in a set of applications, a set of suitable VMs for each application along with the respective job profiles for each machine, and QoS constraints expressed in terms of deadlines for each considered application. Specifically, all these parameters are collected in a JSON file provided as input to the tool.

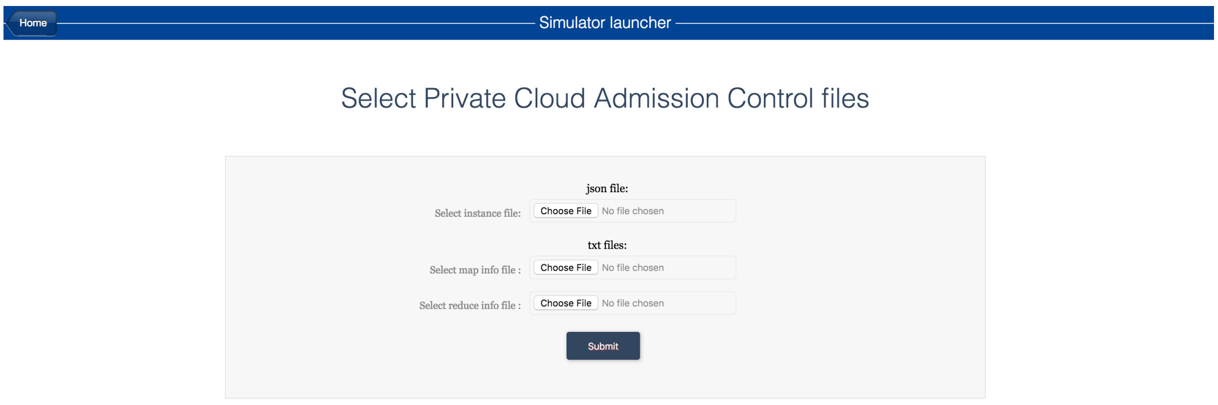
The main components of the backend are the *Initial Solution Builder*, the *Parallel Local Search Optimizer* and the *Evaluator*. Such elements contribute to the identification of an optimized solution for the resource provisioning problem. More details on each component are reported in the following.



The first case helps in the ordinary management of private infrastructure, the objective is to minimize operational costs associated to physical nodes. Admission control is enabled and user concurrency level must be taken into account, thus possibly leading into job rejections if the cluster capacity is not enough to accommodate all the required workload.

In the second case, the interest lies in provisioning a cluster, we do not consider the admission control mechanism here because we are interested in an average workload scenario.

Figure 4: **D-SPACE4Cloud** web GUI: Private Cloud Analysis selection menu



Select Private Cloud Admission Control files

json file:

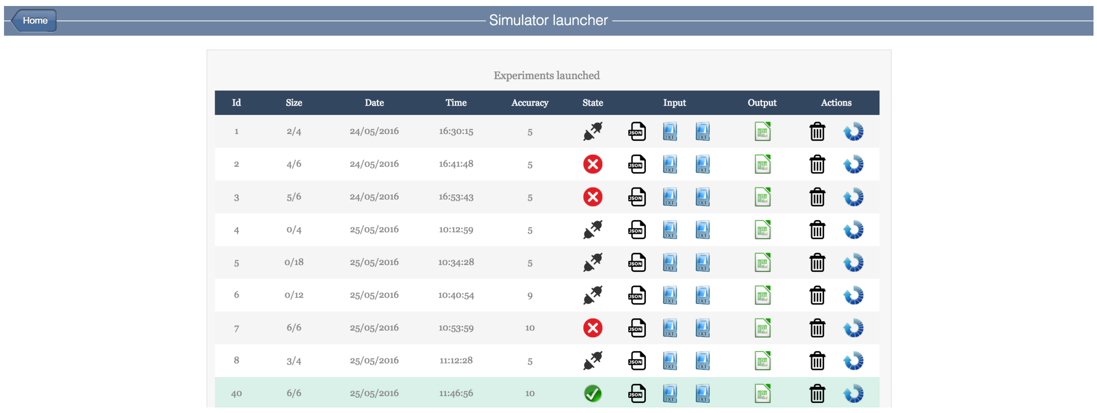
Select instance file: No file chosen

txt files:

Select map info file : No file chosen

Select reduce info file : No file chosen

Figure 5: **D-SPACE4Cloud** web GUI: input form



Experiments launched

Id	Size	Date	Time	Accuracy	State	Input	Output	Actions
1	2/4	24/05/2016	16:30:15	5	🔌	JSON TXT TXT		
2	4/6	24/05/2016	16:41:48	5	✗	JSON TXT TXT		
3	5/6	24/05/2016	16:53:43	5	✗	JSON TXT TXT		
4	0/4	25/05/2016	10:12:59	5	🔌	JSON TXT TXT		
5	0/18	25/05/2016	10:34:28	5	🔌	JSON TXT TXT		
6	0/12	25/05/2016	10:40:54	9	🔌	JSON TXT TXT		
7	6/6	25/05/2016	10:53:59	10	✗	JSON TXT TXT		
8	3/4	25/05/2016	11:12:28	5	🔌	JSON TXT TXT		
40	6/6	25/05/2016	11:46:56	10	✓	JSON TXT TXT		

Figure 6: **D-SPACE4Cloud** web GUI: results page

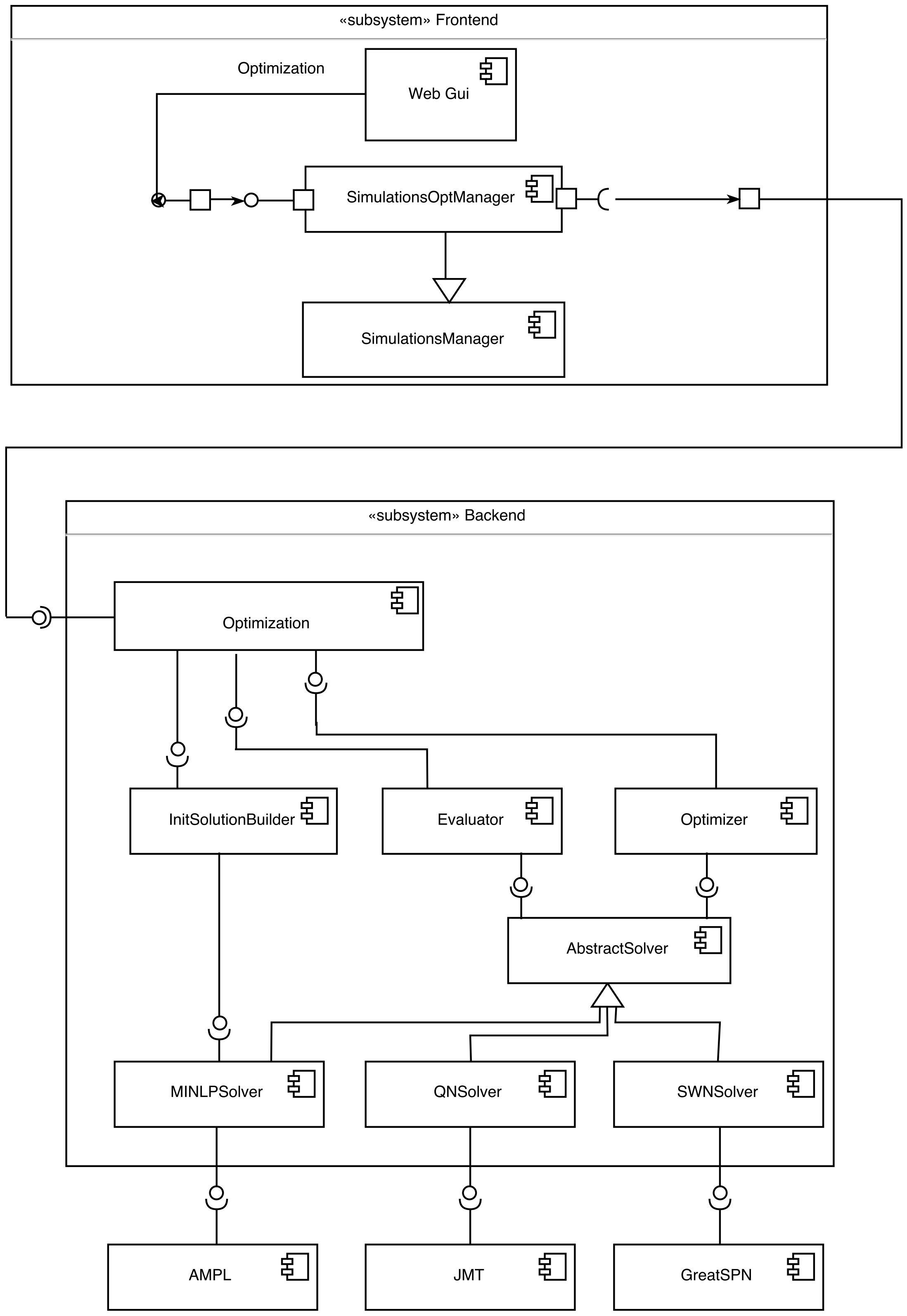


Figure 7: **D-SPACE4Cloud** Component Diagram

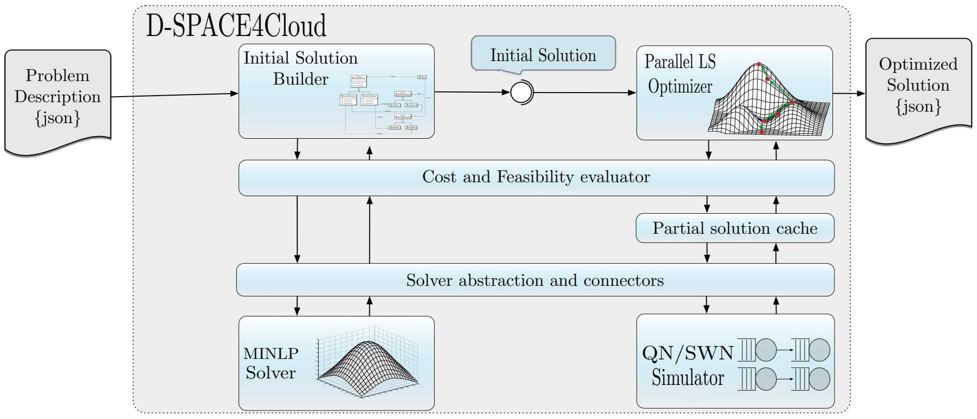


Figure 8: **D-SPACE4Cloud** backend architecture

5.2.1 Initial Solution Builder

The *Initial Solution Builder* generates a starting solution for the problem using a MINLP formulation presented in Appendix B. For more details the reader is referred to [17]. It must be highlighted, at this point, that the quality of the returned solution can still be improved: this is because the MINLP relies on an approximate representation of the Application-Cluster liaison; for this reason the QN model presented in the previous section (or the SWN described in the DICE Deliverable D3.1, *Transformations to analysis models*) is exploited to get a more accurate execution time estimate. The increased accuracy leaves room for further cost reduction; however, since simulation is time consuming, the space of possible cluster configurations has to be explored in the most efficient way, avoiding to evaluate unpromising configurations.

5.2.2 Parallel Local Search Optimizer

In the light of such considerations, 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 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, it is possible to retrieve from the **D-SPACE4Cloud** tool a JSON file listing the results of the overall optimization procedure. 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: refer to Appendix B 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 VMs) is computed so as to minimize the renting out costs of that configuration.

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 ($S_i$)            }  Pursuit of feasibility
 6:             Evaluate ($S_i$)                    }
 7:         end while                               }
 8:     else
 9:         while $S_i$ is feasible do              }
10:             DecrementCluster ($S_i$)            }  Cost optimization
11:             Evaluate ($S_i$)                    }
12:         end while                               }
13:         IncrementCluster ($S_i$)
14:     end if
15: end for
16: return $S$
```

5.2.3 Evaluator

In **D-SPACE4Cloud** the *Evaluator* component carries out the important task of mapping the candidate solutions onto suitable performance models, submit them to the related third-part solver and retrieve the outcomes, namely the mean execution times for each class of jobs. Furthermore, it is in charge of checking the feasibility of evaluated solutions and calculate the related actual cluster leasing costs.

In order to accomplish this task in the most efficient way it implements an abstraction layer and a set of ssh connectors for different solvers (at the time of writing connectors for JMT [3] and GreatSPN [4] are available) that allow to evaluate each solution in parallel (that the mean execution time is estimated for each class of jobs in a parallel and independent way).

Finally, the evaluator embeds a cache system to speed up the evaluation process. As a matter of fact, since a solution is evaluated in parallel and since the local search optimizer may generate similar solutions (i.e., solutions that shares part of their structure), the cache system retains past evaluation and returns them, if needed, in a transparent way. An overall acceleration of the optimization process is the net result of this caching mechanism.

5.3 Third-party services

D-SPACE4Cloud relies on a set of third-party services that are accessed (with the exception of the Resource Database) via SSH connection. Such services are briefly described below:

Resource Database is an open source relational database in MySQL format and **D-SPACE4Cloud** accesses it via JPA. As said, it is an extension of the Resource database realized within MODAClouds EU project and it contains information about several cloud providers. The typical information that can be found and are relevant for the DICE Optimization tool are: name of the provider, virtual resources offered by a certain provider, the memory size and CPU speed for each virtual resource, and the pricing model applied.

MINLP Solver and AMPL AMPL is the acronym for "A Modeling Language for Mathematical Programming" [18]: developed at Bell Laboratories, it is a proprietary algebraic modeling language for linear and nonlinear optimization problems, in discrete or continuous variables. AMPL does not solve problems directly; instead, it communicates with another mathematical software called solver, which is responsible for finding the best solution for the problem. One of the main advantages of AMPL is its syntax: it is very similar to the mathematical optimization problems notation, which makes it very readable and simple to understand. It is available for the most important 32- and 64-bit platforms including Linux, Mac OS X and Windows. There are several solvers available on the market, each developed to solve a single or more problems classes. One of the supported solver is Knitro [19], a tool specifically designed to tackle nonlinear optimization and proven to be effective on a wide range of problems. Internally, it implements several algorithms as the interior and the active-set method along with a set of techniques for the automatic selection of parameters.

GreatSPN and JMT GreatSPN [4] is the tool selected in DICE to solve SWN performance models. In fact, it is able to validate, and evaluate the performance of distributed systems once they have been expressed by means of a specific formal representation; in particular it can solve generalized stochastic Petri nets (GSPNs) and SWNs. State-of-the-art algorithms are implemented within GreatSPN.

Java Modelling Tools (JMT) [3] is a framework developed jointly by Politecnico di Milano and Imperial College London. It is released under the GNU General Public License. Its goal si to offer a comprehensive framework for performance evaluation via analytical and simulation techniques. JMT includes tools for 1. simulating QN models; 2. performing exact and approximate mean value analyses for QN models; 3. performing asymptotic analysis and identifying bottlenecks of QN models; 4. simulating Markov chains.

GreatSPN and JMT can be used alternatively for the evaluation of the solutions obtained via analytical models. JMT boasts shorter simulation times and the replayer feature, through which it is possible to provide service times from measurement data instead of parameterizing a random distribution. On the other hand, GreatSPN can express more general scheduling policies, at the expense of longer simulation times and a narrower range of available distributions.

For further details on GreatSPN and JMT, see DICE Deliverable D1.1 *State of the art analysis*.

6 Validation

In this section we show the results of several experiments performed to validate the proposed approach. All these experiments have been performed on two Ubuntu 14.04 VMs hosted on an Intel Xeon E5530 2.40 GHz equipped server. The first VM ran the **D-SPACE4Cloud** web service and Knitro 10.0 [20], a solver for the mathematical programming language AMPL [5], which was used to solve the optimization problem presented in Appendix B (see [17] for further details), determining an initial solution to our HC algorithm. The second one, instead, ran JMT 0.9.3 [3], a collection of free software performance evaluation programs including a QN simulator and GreatSPN 2.0 [4], which is a performance evaluation tool for GSPNs and SWNs.

Section 6.1 presents the experimental setup and the design of experiments whereas Section 6.2 reports on results of validation experiments of the SWN and QN performance models. Finally, Section 6.3 demonstrates that the optimization approach described in Section 5 is capable of catching realistic behaviors for the system under analysis.

6.1 Experimental Setup and Design of Experiments

In order to obtain job profiles of realistic DIAs, we chose a set of SQL queries, shown in Figure 9, from the industry standard benchmark TPC-DS [21]. We then generated synthetic data compliant with the specifications and executed the queries on Apache Hive [22]. All the selected queries get translated into MapReduce jobs. Notice that we generated data at several scale factors ranging from 250 GB to 1 TB. 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. The numbers of map and reduce tasks varied, respectively, in the ranges $(4, 1560)$ and $(1, 1009)$. These logs are also used to choose a proper distribution with right parameters for the *map* transition in the QN or SWN models.

The parallel execution of multiple tasks within higher level jobs is usually modeled in the QN literature with the concept of fork-join. The performance metrics of such networks must be computed by considering the Markov Chain underlying the QN, due to the lack of a known closed-form solution for fork-join networks with more than two queues. However, such approaches are not fit for Hadoop systems, since the state space grows exponentially with the number of tasks [23, 24]. In particular, [25] proposed a good approximation technique, but it is based on exponentially distributed service times, which is not the case for Hadoop deployments. Our initial experiments showed that mapper and reducer times follow general distributions, which can be approximated by Markovian arrival processes or in some cases Erlang [26]: in particular, we used Erlang-2 for R1, Erlang-4 for R2 and R3, and Erlang-5 for R4 and R5. Further details about distributions can be found in deliverable D4.5 *Iterative quality enhancement tools - Initial version*.

Profiling has been performed on Amazon EC2, by considering m4.xlarge instances, and on PICO,[6] 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. In the first case every container had 2 GB RAM and in the second 6 GB. In the end, we recorded the different VM types characteristics.

6.2 Performance Models Validation

To start off with, we show results for the validation of the SWN and QN models. We feed them with parameters evaluated on the real systems we took into account and compare the measured performance metrics with the ones obtained via simulation. Specifically, we consider as a quality index the accuracy on the prediction of response times, defined as follows:

$$\vartheta = \frac{\tau - T}{T} \tag{1}$$

[6] `http://www.hpc.cineca.it/hardware/pico`

```sql
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

```sql
select inv_item_sk,inv_warehouse_sk
from inventory where inv_quantity_on_hand > 10
group by inv_item_sk,inv_warehouse_sk
having sum(inv_quantity_on_hand)>20
limit 100;
```

(b) R2

```sql
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;
```

(c) R3

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

(d) R4

```sql
select inv_warehouse_sk, sum(inv_quantity_on_hand)
from inventory
group by inv_warehouse_sk
having sum(inv_quantity_on_hand) > 5
limit 100;
```

(e) R5

Figure 9: Queries

Table 1: QN and SWN models accuracy

Query	Users	Cores	Scale [GB]	n^M	n^R	T [ms]	τ_{QN} [ms]	ϑ_{QN} [%]	τ_{SWN} [ms]	ϑ_{SWN} [%]
R1	1	240	250	500	1	55410	50753.34	−8.40	50629.58	−8.63
R2	1	240	250	65	5	36881	27495.31	−25.45	37976.82	2.97
R3	1	240	250	750	1	76806	77260.03	0.60	83317.27	8.48
R4	1	240	250	524	384	92197	78573.96	−14.72	89426.51	−3.01
R1	1	60	500	287	300	378127	411940.93	8.94	330149.74	−12.69
R3	1	100	500	757	793	401827	524759.36	30.59	507758.68	26.36
R3	1	120	750	1148	1009	661214	759230.77	14.82	698276.75	5.61
R4	1	60	750	868	910	808490	844700.85	4.48	806366.51	−0.26
R3	1	80	1000	1560	1009	1019973	1053829.78	−1.00	1020294.84	0.03
R5	1	80	1000	64	68	39206	36598.32	−6.65	38796.47	−1.04
R1	3	20	250	144	151	1002160	1038951.05	3.67	909217.89	−9.27
R1	5	20	250	144	151	1736949	1215490.20	−30.02	1428894.40	−17.74
R2	3	20	250	4	4	95403	112050.45	17.45	99219.94	4.00
R2	5	20	250	4	4	145646	97619.46	−32.97	88683.10	3.09
R1	5	40	250	144	151	636694	660241.29	3.70	613577.53	−3.63
R2	3	40	250	4	4	86023	105785.41	22.97	119712.30	−17.81
R2	5	40	250	4	4	90674	103173.38	13.78	117582.82	29.68

where τ is the simulated response time, whilst T is the average measured one. Such a definition allows not only to quantify the relative error on response times, but also to identify cases where the predicted time is smaller than the actual one, thus leading to possible deadline misses. Indeed, if $\vartheta < 0$ then the prediction is not conservative.

Among these experiments, we considered both single user scenarios, where one query has been run repeatedly on dedicated cluster, interleaving a 10 s average think time between completions and subsequent submissions, and multiple user scenarios, with several users concurrently interacting with the cluster in a similar way.

Table 1 shows the results of the QN and SWN models validation. For all the experiments we report the number of concurrent users, the overall cores available in the cluster, the dataset scale factor, and the total number of map and reduce tasks, plus the above mentioned metric. In the worst case, the relative error can reach up to 32.97%, which is in line with the expected accuracy in the performance prediction field [14]. Moreover, the SWN model achieves a higher accuracy, with the average relative error decreasing from the 14.13% of QNs down to 9.08%. For these experiments, we considered both single user scenarios, repeatedly running the same query on a dedicated cluster with $Z_i = 10$ s, and multiple users scenarios.

6.3 Scenario-based Experiments

In this section we show that the optimization approach described in Section 5 is capable of catching realistic behaviors for the system under analysis. We test this property with a set of 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 clusters hosted on public Clouds are the level of concurrency and the deadlines. In the first case, increasing the number of concurrent users 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, 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 10, 11 and 12 report the solutions obtained with the 250 GB dataset profiles. The average running time for these experiments is about two hours. All the mentioned figures show the per hour cost plotted against decreasing deadlines in ms for both the real VM types considered: CINECA is the 20-core node available on PICO, whilst m4.xlarge is the 4-core instance rented on Amazon AWS. In Figures 10 and 11 the expected cost increase due to tightening deadlines is apparent for both query R1 and R3, considering 10 concurrent users. Further, in both cases it is cheaper to provision a Cloud cluster consisting of the smaller Amazon-offered instances, independently of the deadlines. It is then interesting to observe that R1 shows a different behavior if the required concurrency level increases. Figure 12 shows that, as the deadlines become tighter and tighter, it is possible to identify a region where executing the workload on larger VMs becomes more economic.

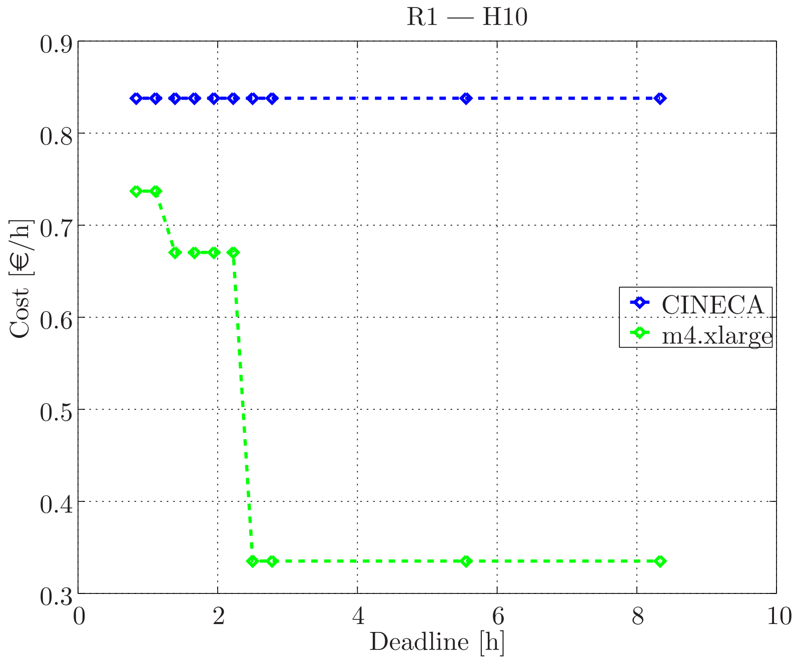


Figure 10: Query R1, 10 concurrent users

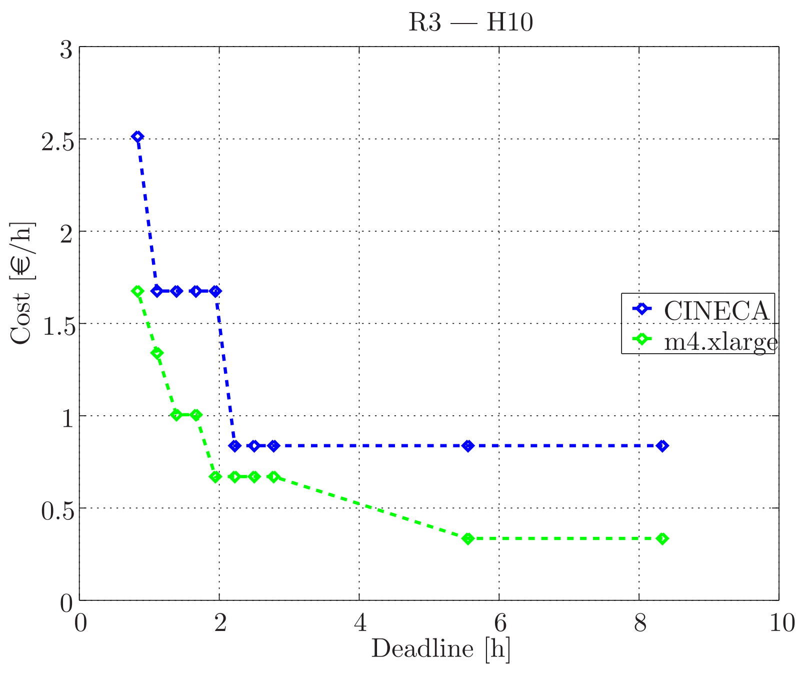


Figure 11: Query R3, 10 concurrent users

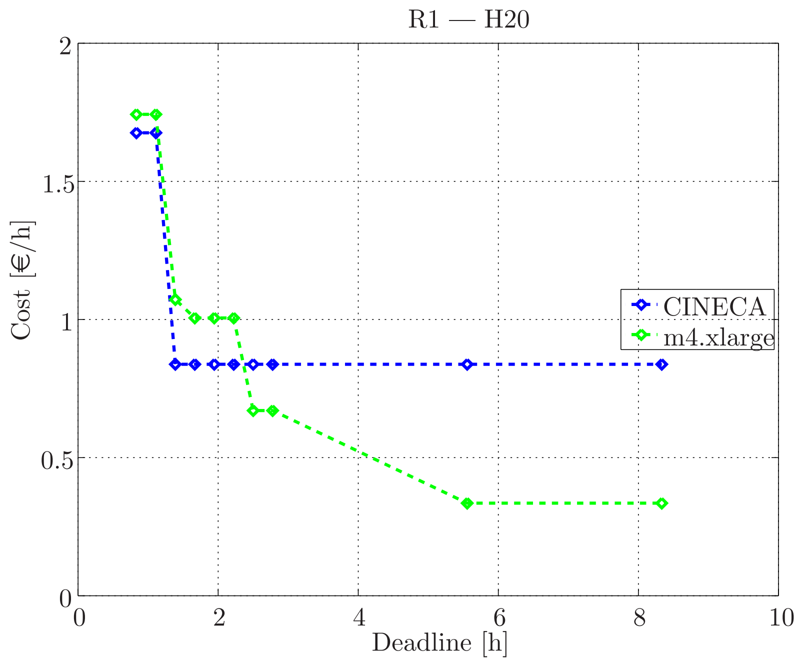


Figure 12: Query R1, 20 concurrent users

7 Conclusions and future works

In this section we provide a wrap-up of what has been accomplished so far with the development of the DICE Optimization framework.

The main achievements of this deliverable in relation to the initial requirements for the tool are shown in Table 2. The primary focus of our activities was on developing architectures and algorithms for the optimization on Map-Reduce based DIAs (**R3.8**). As a side activity fast-to-solve QN models have been devised and validated. We implemented the DICE Optimization tool as an independent distributed, and web-based software system; intermediate json files and embedded performance models have been used since DDSM diagrams and M2M APIs are currently under completion.

Currently our tool can be used as a standalone application and provides a graphical interface to launch optimization experiment in batch and in parallel exploiting a SOA-based architecture and hiding the complexity of third-party solvers/simulators.

Requirement ID	Description	Coverage	To do
R3.8	Cost/quality balance	60 %	Spark and Storm cases must be considered: complete Hadoop and start working on them at M24, finalize at M30
R3.10	SLA specification and compliance	80 %	Integration with the IDE (M30)
R3.11	Optimization timeout	0 %	To be done

Table 2: Requirement coverage at month 18.

7.1 Further work

Starting from the requirements listed in Table 2, the following items provide an overview of the next issues to be addressed within Task T3.4 and of the forthcoming work that will be carried out until M36.

R3.8. In the next periods new algorithms and models will be proposed to assess and optimize the development of Spark and Storm applications. The Backend service will be modified leaving its general architecture unchanged; nonetheless, the initial solution generation process and the local search must be adjusted to consider the peculiarities of those technologies.

R3.10. SLA details are specified by the ARCHITECT using the DICE profile within the DICE IDE. Currently there is no mechanism that transform DICE profile, and in particular the SLA, to **D-SPACE4Cloud** internal format and vice versa. We will address this issue with the help of the team developing the IDE.

R3.11. Currently, the DICE simulation tools do not provide suitable APIs to set timeouts for external tools. This has impeded the implementation of the optimization timeout in DICE Optimization tool. However, we are preparing the software in order to make the implementation of such feature as painless as possible once it will be available at the DICE Simulation tools.

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] *AMPL*. URL: http://www.ampl.com/ (visited on 03/09/2016).

[6] *CMPL 1.11.0*. URL: http://www.coliop.org (visited on 07/19/2016).

[7] *GLPK (GNU Linear Programming Kit)*. URL: https://www.gnu.org/software/glpk/ (visited on 07/19/2016).

[8] *Apache Hadoop*. URL: http://hadoop.apache.org (visited on 11/17/2015).

[9] *Amazon Elastic MapReduce*. URL: https://aws.amazon.com/elasticmapreduce/ (visited on 08/30/2015).

[10] *Microsoft HDInsight*. URL: http://azure.microsoft.com/en-us/services/hdinsight/ (visited on 08/30/2015).

[11] *Amazon Simple Storage Service*. URL: https://aws.amazon.com/s3/ (visited on 09/17/2015).

[12] *Microsoft Azure Storage*. URL: http://azure.microsoft.com/en-us/services/storage/ (visited on 09/17/2015).

[13] *Amazon EC2 Pricing*. URL: http://aws.amazon.com/ec2/pricing/ (visited on 07/16/2015).

[14] 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 04/07/2015).

[15] 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.

[16] 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.

[17] 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.

[18] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. 2nd ed. Cengage Learning, 2002. ISBN: 0534388094.

[19] Richard H Byrd, Jorge Nocedal, and Richard A Waltz. "Knitro: An Integrated Package for Nonlinear Optimization". In: *Energy*. Vol. 83. 2006, pp. 35–59. ISBN: 978-0-387-30065-8. DOI: 10.1007/0-387-30065-1_4. arXiv: arXiv:1011.1669v3. URL: http://www.ziena.com/papers/integratedpackage.pdf$%5Cbackslash$nhttp://link.springer.com/10.1007/0-387-30065-1%7B%5C_%7D4.

[20] *Artelys Knitro*. URL: `http://www.artelys.com/en/optimization-tools/knitro` (visited on 03/09/2016).

[21] *TPC-DS Benchmark*. URL: `http://www.tpc.org/tpcds/` (visited on 03/09/2016).

[22] *Apache Hive*. URL: `https://hive.apache.org` (visited on 03/09/2016).

[23] Wesley W. Chu, Chi-Man Sit, and Kin K. Leung. “Task Response Time for Real-Time Distributed Systems with Resource Contentions”. In: *IEEE Trans. Softw. Eng.* 17.10 (), pp. 1076–1092. ISSN: 0098-5589. DOI: `10.1109/32.99195`.

[24] V. W. Mak and S. F. Lundstrom. “Predicting Performance of Parallel Computations”. In: *IEEE Trans. Parallel Distrib. Syst.* 1.3 (July 1990), pp. 257–270. ISSN: 1045-9219. DOI: `10.1109/71.80155`.

[25] Randolf D. Nelson and Asser N. Tantawi. “Approximate Analysis of Fork/Join Synchronization in Parallel Queues”. In: *IEEE Trans. Computers* 37.6 (1988), pp. 739–743. DOI: `10.1109/12.2213`.

[26] The DICE Consortium. *Design and Quality Abstractions — Initial Version*. 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/75/2016/02/D2.1_Design-and-quality-abstractions-Initial-version.pdf`.

[27] 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.

[28] 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.

Listing 1: Example back end properties file

```
spring.profiles.active = test
file-management.deletion-policy = delete
solver.type = QNSolver
server.port = 8081

minlp.address = your.minlp.server.org
minlp.username = username
minlp.port = 22
minlp.remote-work-dir = /home/username/AMPL
minlp.ampl-directory = ampl
minlp.solver-path = knitroampl
minlp.known-hosts = ${HOME}/.ssh/known_hosts
minlp.private-key-file = ${HOME}/.ssh/id_rsa
minlp.force-clean = false

SPN.address = your.greatspn.server.org
SPN.username = username
SPN.port = 22
SPN.solver-path = swn_ord_sim
SPN.remote-work-dir = /home/username/GreatSPN
SPN.accuracy = 10
SPN.known-hosts = ${HOME}/.ssh/known_hosts
SPN.private-key-file = ${HOME}/.ssh/id_rsa
SPN.force-clean = false

QN.address = your.jmt.server.org
QN.username = username
QN.port = 22
QN.model = class-switch
QN.solver-path = /home/username/JavaModellingTools/JMT.jar
QN.remote-work-dir = /home/username/JMT
QN.accuracy = 10
QN.significance = 0.05
QN.known-hosts = ${HOME}/.ssh/known_hosts
QN.private-key-file = ${HOME}/.ssh/id_rsa
QN.force-clean = false
QN.max-duration = 7200

s4c.parallel = true

logging.file = log.txt
```

A Installation and Usage Manual

This appendix provides a user guide for the deployment and usage of **D-SPACE4Cloud**. Currently we do not provide precompiled binary packages, hence Maven and a Java 8 JDK need to be installed.

The first step is to download the back end.[7] To compile it, move to the root of the downloaded repository and run `mvn package`. Under `target/` you will find the relocatable jar file. Create a folder to host the web service, copy the jar file, and add a property file, see Listing 1. As soon as your configuration is in place, you can launch the back end with `java -jar <filename.jar>`. Most likely, you should consider using `nohup`, GNU Screen, or similar software, in order to have the web service survive your session. Repeat these steps to obtain a compiled archive of the web front end.[8] An example properties file is shown in Listing 2.

You should bear in mind that the solver paths in the back end configuration should be either command names available in the remote system `PATH` or absolute paths to the solver executables. Moreover, the connection with solvers and simulators is established via SSH, hence you should provide an address and port where the remote SSH daemon listens, the path to your local `known_hosts` file, the remote user name, and the path to an authorized private key file. In the end, the `accuracy` and `significance` properties can be used to tune the stopping criterion observed by the solvers.

The ports listed in the front end `launcher.ports` property must be those configured in the back

[7] `https://github.com/deib-polimi/diceH2020-space4cloudsWS`
[8] `https://github.com/deib-polimi/diceH2020-space4cloud-webGUI`

Listing 2: Example front end properties file

```
launcher.instance-dir = instances
launcher.txt-dir = instances
launcher.sol-instance-dir = solInstances
launcher.result-dir = results
launcher.address = your.back.end.server.org
launcher.ports = 8081,8082
server.port = ${port:8080}

logging.file = logLauncher.txt

spring.mail.username = name.surname@gmail.com
spring.mail.password = clear-text-password
spring.mail.host = smtp.gmail.com
spring.mail.port = 587

email.enabled = true
email.recipients = name.surname@gmail.com,your.boss@gmail.com
```

end files as `server.port`. Currently you can use multiple back ends at once, provided they are deployed on the same server. In addition, it is possible to configure the front end to send a notification email to a list of recipients when major events occur, for instance when a batch of optimizations completes.

Both the back and the front end have a configurable deletion policy for debugging purposes. The default setting for the `file-management.deletion-policy` property is `delete`, meaning that every temporary file gets deleted as soon as it is not needed anymore. The other two possible values are `delete-on-exit`, with files that persist until the application closes, and `keep-files`, which never deletes any temporary.

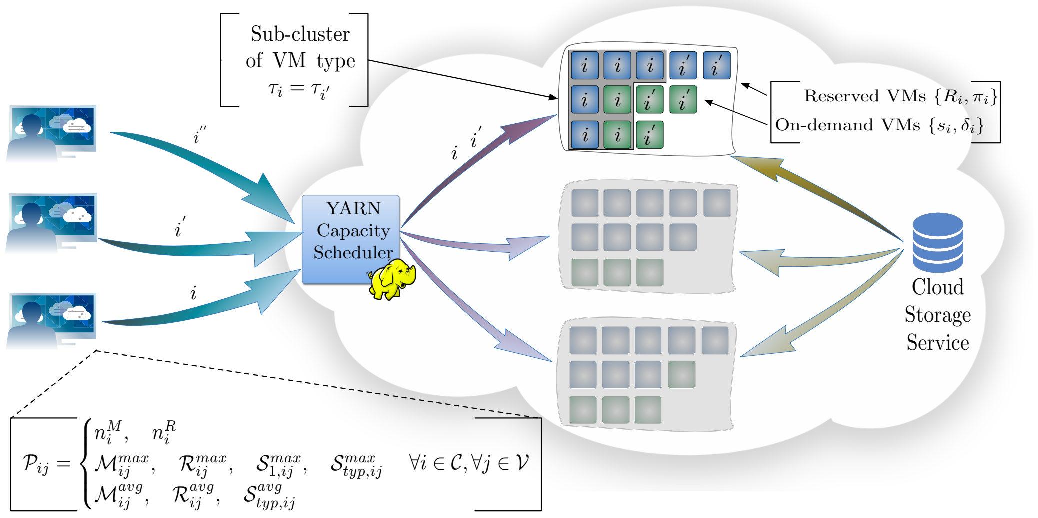

Sub-cluster of VM type $\tau_i = \tau_{i'}$
Reserved VMs $\{R_i, \pi_i\}$
On-demand VMs $\{s_i, \delta_i\}$
i''
i'
i
YARN Capacity Scheduler
i i'
Cloud Storage Service
$\mathcal{P}_{ij} = \begin{cases} n_i^M, \quad n_i^R \\ \mathcal{M}_{ij}^{max}, \quad \mathcal{R}_{ij}^{max}, \quad \mathcal{S}_{1,ij}^{max}, \quad \mathcal{S}_{typ,ij}^{max} \quad \forall i \in \mathcal{C}, \forall j \in \mathcal{V} \\ \mathcal{M}_{ij}^{avg}, \quad \mathcal{R}_{ij}^{avg}, \quad \mathcal{S}_{typ,ij}^{avg} \end{cases}$

Figure 13: Reference system

B Mathematical Formulation for the Resource Provisioning problem

In this section we summarize the main concepts introduced in Section 4.2 and we present the notation required to formulate the optimization of the deployment configuration as an optimization problem.

We envision the following scenario, wherein a company needs to set up a cluster to carry out efficiently a set of interactive MapReduce jobs. Different classes $\mathcal{C} = \{i \,|\, i = 1, \ldots, n\}$ gather MapReduce jobs that show a similar behavior.

A Hadoop 2.x cluster featuring the YARN Capacity Scheduler and running on a public Cloud IaaS is considered. 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.

Moreover, YARN is configured in a way that all available cores can be dynamically assigned to either Map or Reduce tasks. Finally, in order to limit the risk of data corruption and according to the practices suggested by major Cloud vendors [9, 10], the datasets reside on an external storage infrastructure [11, 12] accessible at quasi-constant time.

As, in general, IaaS (IaaS) providers feature a limited, but possibly large, 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. We denote with τ_i the VM type j used to support jobs of class i and with ν_i the number of VMs of such a kind allocated to class i. In this scenario, we consider a pricing model derived from *Amazon EC2* [13]. The provider offers: 1) *reserved* VMs, for which it adopts a one-time payment policy that grants access to a certain number of them for the contract duration; and 2) *on-demand* VMs, that customers can buy at an higher price to compensate peaks in the incoming workload. In order to obtain the most cost-effective configuration, we rely on reserved VMs (r_i) for the bulk of computational needs and complement them with on-demand VMs (d_i). Being ν the total number of VMs leased, it can be expressed as: $\nu_i = r_i + d_i$. Let δ_{τ_i} be the unit cost for on-demand VMs of type τ_i, whilst π_{τ_i} is the effective hourly cost for one reserved VM: it is the unit upfront payment normalized over the contract duration. Overall, the cluster hourly renting out costs can be calculated as follows:

$$\text{cost} = \sum_{i \in \mathcal{C}} \left(\delta_{\tau_i} d_i + \pi_{\tau_i} r_i \right) \tag{2}$$

Reducing the operating costs of the cluster by using efficiently the leased virtual resources is in the interest of the company. This translates into a Resource Provisioning problem where the renting out costs must be minimized subject to the fulfillment of QoS requirements, namely H_i per-class concurrency level given certain deadlines D_i. In the following we assume that the system supports H_i users for each class and that users work interactively with the system and run another job after a think time exponentially distributed with mean Z_i, i.e., the system is represented as a closed model [14].

In order to rigorously 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 approach presented in [15] it is possible to derive from the Hadoop logs a *job profile*, that is a concise behavior characterization for each class. Following the notation brought forth in [15, 27], given a certain VM of type j, the job profile $\mathcal{P}_{ij}$ for application class i aggregates the following information: 1) n_i^M and n_i^R, respectively the total number of Map and Reduce tasks per job; 2) $\mathcal{M}_{ij}^{max}$, $\mathcal{R}_{ij}^{max}$, $\mathcal{S}_{1,ij}^{max}$, and $\mathcal{S}_{typ,ij}^{max}$, the maximum duration of a single Map, Reduce, and Shuffle task (notice that the first Shuffle wave of a given job is distinguished from all the subsequent ones); 3) $\mathcal{M}_{ij}^{avg}$, $\mathcal{R}_{ij}^{avg}$, and $\mathcal{S}_{typ,ij}^{avg}$, i.e., the average duration of Map, Reduce, and Shuffle tasks, respectively.

Given the amount and type of resources allocated, the concurrency level, and the job profile, the estimated execution time can generically be expressed as in (3):

$$T_i = \mathcal{T}\left(\mathcal{P}_{i,\tau_i}, \nu_i; H_i, Z_i\right), \quad \forall i \in \mathcal{C}. \tag{3}$$

What is worthwhile to note is that the previous formula represents a general relation describing either closed form results, as those presented in [27], or the average execution times derived via simulation, the approach adopted in this deliverable. Since the execution of jobs on a suboptimal VM type might give rise to performance disruptions, it is critical to avoid assigning tasks belonging to class i to the wrong VM type $j \neq \tau_i$. Indeed, YARN allows for specifying Node Labels and partitioning nodes in the cluster according to these labels, then it is possible to enforce this separation. Our configuration 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, where idle resources can be assigned to jobs requiring the same VM 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. Equations (3) can be used to formulate the deadline constraints as:

$$T_i \leq D_i, \quad \forall i \in \mathcal{C}. \tag{4}$$

In light of the above, we can 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 VMs $\nu_i = r_i + d_i$ 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 13 for a graphical overview of the main elements of the considered resource provisioning problem. Furthermore, in Table 3 a complete list of the parameters used in the models presented in the next sections is reported, whilst Table 4 summarizes the decision variables.

In the following we present the optimization model and techniques exploited by the **D-SPACE4Cloud** tool in order to determine the optimal VM mix given the profiles characterizing the applications under study and the possible Cloud provider to host the virtual cluster.

Basic building blocks for the optimization tool are the models of the system under study. First of all, we need a quick, although rough, method to estimate completion times and operational costs: to this end, we exploit a mathematical programming formulation. In this way, 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 demanding simulations, reaching a precise 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 their price and the expected performance. This will be done by exploiting a set of logical variables x_{ij}: we will enforce that only $x_{i,\tau_i} = 1$, thus

Table 3: Model parameters

Parameter	Definition
$\mathcal{C}$	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
δ_j	Unit hourly cost for on demand VMs of type j [€/h]
π_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

Table 4: Decision variables

Variable	Definition
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
x_{ij}	Binary variable equal to 1 if class i is hosted on VM type j

determining the optimal VM type τ_i for application class i. We address this issue proposing the following mathematical programming formulation:

$$\min_{\mathbf{x},\boldsymbol{\nu},\mathbf{R}} \quad \sum_{i\in\mathcal{C}} \left(\delta_{\tau_i} d_i + \pi_{\tau_i} r_i\right) \tag{P1a}$$

subject to:

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

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

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

$$\pi_{\tau_i} = \sum_{j\in\mathcal{V}} \pi_j x_{ij}, \quad \forall i \in \mathcal{C} \tag{P1e}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in \mathcal{C}, \forall j \in \mathcal{V} \tag{P1f}$$

$$(\boldsymbol{\nu}, \mathbf{R}) \in \arg\min \sum_{i\in\mathcal{C}} \left(\delta_{\tau_i} d_i + \pi_{\tau_i} r_i\right) \tag{P1g}$$

subject to:

$$r_i \le R_{i,\tau_i}, \quad \forall i \in \mathcal{C} \tag{P1h}$$

$$\nu_i = r_i + d_i, \quad \forall i \in \mathcal{C} \tag{P1i}$$

$$\mathcal{T}\left(\mathcal{P}_{i,\tau_i}, \nu_i; H_i, Z_i\right) \le D_i, \quad \forall i \in \mathcal{C} \tag{P1j}$$

$$\nu_i \in \mathbb{N}, \quad \forall i \in \mathcal{C} \tag{P1k}$$

$$r_i \in \mathbb{N}, \quad \forall i \in \mathcal{C} \tag{P1l}$$

$$d_i \in \mathbb{N}, \quad \forall i \in \mathcal{C} \tag{P1m}$$

(P1n)

Problem (P1) is a bilevel 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 (P1e), pick the values for the inner problem parameters.

The inner objective function (P1g) has the same expression as (P1a), but in this case the prices δ_{τ_i} and π_{τ_i} are fixed, as they have been chosen at the upper level. Constraints (P1h) 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, (P1i) add all the VMs available for class i, irrespective of the pricing model. Further, constraints (P1j) mandate to respect the deadlines D_i, as stated in the SLA contracts. In the end, all the remaining decision variables are taken from the natural numbers set, according to their interpretation.

The presented formulation of Problem (P1) 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 [28], integer programming problems belong to the NP-hard class, hence the same applies to (P1). However, since there is no constraint linking variables belonging to different application classes, we can split this general formulation into several smaller and independent problems, one per class i.

C Developers' Documentation

In this appendix we provide detailed documentation for developers' use.

C.1 D-SPACE4Cloud Back End REST APIs

The base URL for all the endpoints is composed of:

- `address`: the address of the **D-SPACE4Cloud** back end;
- `port`: the port where the **D-SPACE4Cloud** back end can be accessed.

Note that the port is configurable through the `application.properties` file of the web service.

Quick Reference

Table 5 report a short description of the REST APIs exposed by the **D-SPACE4Cloud** back end. They are accessible over HTTP at the base domain `http://address:port`.

Table 5: REST APIs exposed by the **D-SPACE4Cloud** back end web service

URL	HTTP verb	Functionality
`/event`	POST	trigger state machine transition
`/state`	GET	retrieve current state
`/upload`	POST	upload text files needed for the replayer
`/inputdata`	POST	upload optimization input JSON file
`/solution`	GET	retrieving final solution
`/solution`	POST	upload solution JSON for assessment analysis
`/settings`	POST	setup accuracy, simulation duration, solver

Please notice that each POST HTTP method returns as response a string reporting the state of the back end. Furthermore, the **`/event`** endpoint is useful to launch optimizations. If, after the transition, the web service arrives in state:

RUNNING_INIT optimization will be run on the received `InstanceData`;

EVALUATING_INIT duration, cost, and feasibility of the received `Solution` will be calculated;

RUNNING_LS hill climbing will be executed on the received `Solution`.

C.2 Finite-State Machine

Every **D-SPACE4Cloud** back end web service keeps track of its own state via an embedded FSM, as shown in Figure 14. As soon as the web service launches, the FSM attains the STARTING state, which lasts until all the initialization tasks have been carried out. If anything prevents the web service from initializing properly, the *STOP* transition sets the ERROR state. Otherwise, the back end *MIGRATE*s to IDLE.

IDLE is the state where the web service waits for input. Generally, unrecoverable issues make the back end *STOP*, moving to the ERROR state. On the other hand, the FSM may return IDLE either forcibly, via *RESET*, or as part of the normal life-cycle, with *MIGRATE*. When a `Solution` is ready for retrieval, the FSM reaches the FINISH state. The reader is referred to Section C.3 for a correspondence between FSM states and phases of the application workflow.

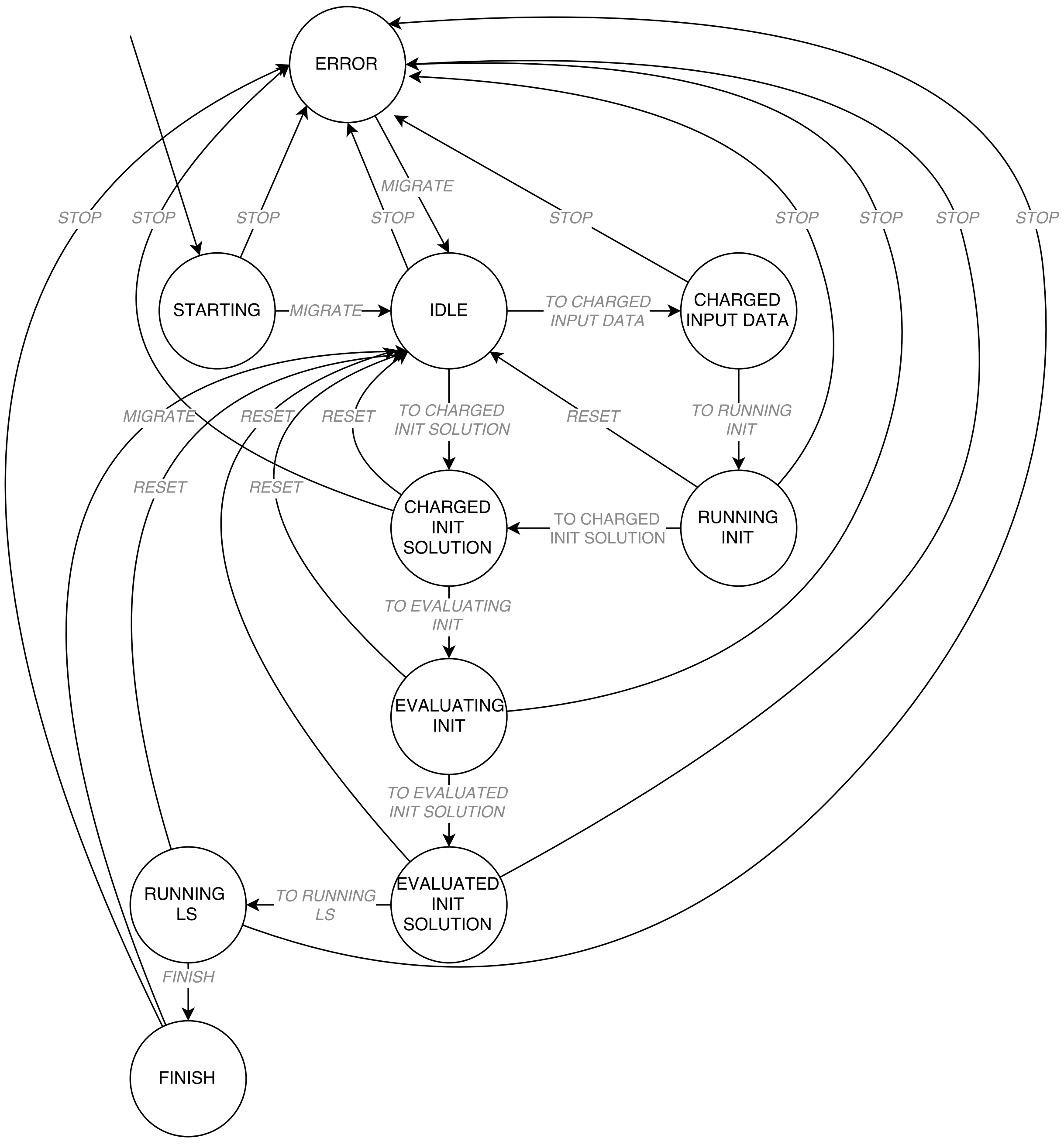


Figure 14: FSM implemented within the **D-SPACE4Cloud** back end service

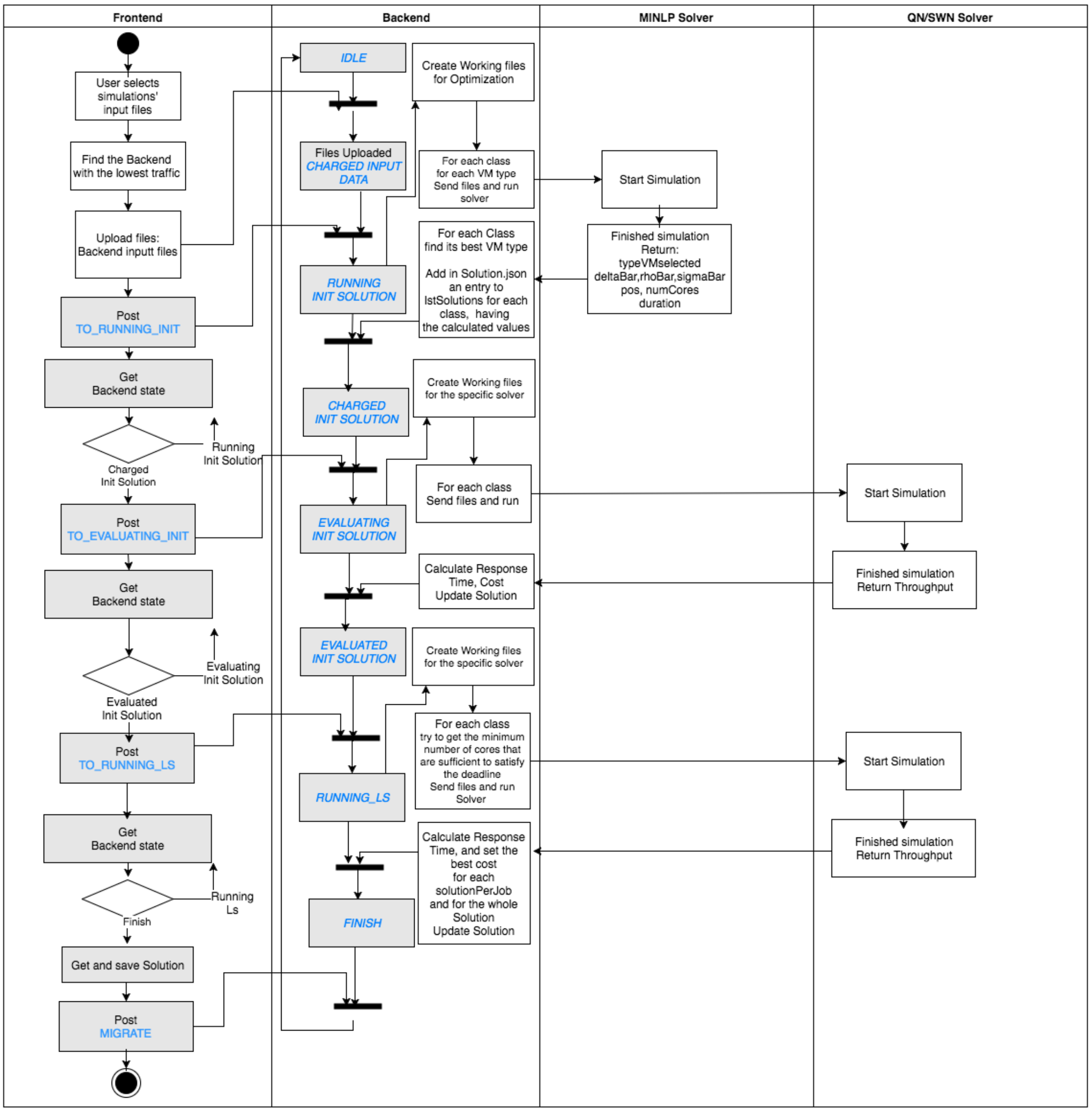

Frontend
Backend
MINLP Solver
QN/SWN Solver
User selects simulations' input files
Find the Backend with the lowest traffic
Upload files: Backend inputt files
Post TO_RUNNING_INIT
Get Backend state
Running Init Solution
Charged Init Solution
Post TO_EVALUATING_INIT
Get Backend state
Evaluating Init Solution
Evaluated Init Solution
Post TO_RUNNING_LS
Get Backend state
Running Ls
Finish
Get and save Solution
Post MIGRATE
IDLE
Files Uploaded CHARGED INPUT DATA
RUNNING INIT SOLUTION
CHARGED INIT SOLUTION
EVALUATING INIT SOLUTION
EVALUATED INIT SOLUTION
RUNNING_LS
FINISH
Create Working files for Optimization
For each class for each VM type Send files and run solver
For each Class find its best VM type Add in Solution.json an entry to lstSolutions for each class, having the calculated values
Create Working files for the specific solver
For each class Send files and run
Calculate Response Time, Cost Update Solution
Create Working files for the specific solver
For each class try to get the minimum number of cores that are sufficient to satisfy the deadline Send files and run Solver
Calculate Response Time, and set the best cost for each solutionPerJob and for the whole Solution Update Solution
Start Simulation
Finished simulation Return: typeVMselected deltaBar,rhoBar,sigmaBar pos, numCores duration
Start Simulation
Finished simulation Return Throughput
Start Simulation
Finished simulation Return Throughput

Figure 15: Optimization activity diagram

C.3 Optimization Activity Diagram

Figure 15 shows in detail the activities carried out by the whole **D-SPACE4Cloud** architecture. In the beginning, the user uploads the required input files: an example is available in Section C.4. The front end, then, looks for the least loaded back end to transmit the relevant data. As soon as the upload is over, the GUI sends *TO RUNNING INIT*, thus triggering the optimization algorithm. The back end works asynchronously, hence the front end just needs to keep track of the FSM evolution and retrieve the `Solution` when the optimization procedure completes.

Meanwhile, the web service chooses provider and VM type according to the results of the MINLP problem discussed in Appendix B. When the initial solution is ready, the backend relies on either the QN or SWN simulator to evaluate it: depending on these results, the following local search procedure will aim at either meeting the prearranged deadline or saving on the running costs. According to the preliminary evaluation, the backend applies HC iteratively calling the configured simulator after every move: as soon as the stopping criterion is satisfied, the FSM reaches the FINISH state and the `Solution` is available for retrieval.

C.4 Input Files

This section discusses examples of the required input files. All the time measures are expressed in ms, but this is not a strict requirement: they only need be coherent.

Listing 3 is the main input file. It collects all the relevant profiles, organized by application class and by provider. For all the class-provider combinations, the JSON has a map from VM type to job profile, since computational capabilities affect the obtained performance. In the example file, we have the R1 and R2 queries, with the profiles obtained on different Amazon EC2 instances. On the other hand, Listing 4 contains parameters characterizing each application class, specifically those related to QoS. In the example JSON you can read the deadline `d`, the penalty associated to the rejection of one job, the expected (`hup`) and minimal (`hlow`) required concurrency level, in addition to the requirements in terms of CPUs (`v`) and RAM (`m`) of each container.

Listing 3: Example JSON file with job profiles

```json
{
  "R1": {
    "Amazon": {
      "large": {
        "cm": 4,
        "cr": 4,
        "mavg": 23840,
        "mmax": 30723,
        "nm": 4,
        "nr": 4,
        "ravg": 2352,
        "rmax": 2841,
        "sh1max": 0,
        "shtypavg": 5381,
        "shtypmax": 6501
      },
      "medium": {
        "cm": 2,
        "cr": 2,
        "mavg": 19040,
        "mmax": 25923,
        "nm": 4,
        "nr": 4,
        "ravg": 2682,
        "rmax": 3171,
        "sh1max": 0,
        "shtypavg": 6136,
        "shtypmax": 7257
      },
      "xlarge": {
        "cm": 8,
        "cr": 8,
        "mavg": 28640,
        "mmax": 35523,
        "nm": 4,
        "nr": 4,
        "ravg": 2022,
        "rmax": 2511,
        "sh1max": 0,
        "shtypavg": 4626,
        "shtypmax": 5746
      }
    }
  },
  "R2": {
    "Amazon": {
      "large": {
        "cm": 4,
        "cr": 4,
        "mavg": 26624,
        "mmax": 39586,
        "nm": 144,
        "nr": 151,
        "ravg": 2981,
        "rmax": 4509,
        "sh1max": 0,
        "shtypavg": 6821,
        "shtypmax": 10316
      },
      "medium": {
        "cm": 2,
        "cr": 2,
        "mavg": 21024,
        "mmax": 33986,
        "nm": 144,
        "nr": 151,
        "ravg": 3408,
        "rmax": 4936,
        "sh1max": 0,
        "shtypavg": 7799,
        "shtypmax": 11294
      },
      "xlarge": {
        "cm": 8,
        "cr": 8,
        "mavg": 32224,
        "mmax": 45186,
        "nm": 144,
        "nr": 151,
        "ravg": 2553,
        "rmax": 4081,
        "sh1max": 0,
        "shtypavg": 5843,
        "shtypmax": 9338
      }
    }
  }
}
```

Whilst Listings 3 and 4 characterize jobs, thus being needed in every optimization scenario, there are also additional JSON files that may or may not be required depending on the particular optimization task at hand, as discussed in Section 3.2. For instance, the parameters characterizing theVM types offered by various public cloud providers are embedded in the back end database, but this is not the case for

Listing 4: Example JSON file with class parameters

```
{
  "R1": {
    "d": 3600000,
    "hlow": 7,
    "hup": 10,
    "id": "R1",
    "job_penalty": 35,
    "m": 2,
    "think": 10000,
    "v": 1
  },
  "R2": {
    "d": 3600000,
    "hlow": 7,
    "hup": 10,
    "id": "R2",
    "job_penalty": 28,
    "m": 4,
    "think": 10000,
    "v": 2
  }
}
```

alternatives available on private infrastructures: Listing 5 provides their characteristics. Furthermore, Listing 6 lists the parameters characterizing the homogeneous nodes that compose the physical cluster: in particular, the example cluster features `N` homogeneous nodes with `V` CPUs, `M` GB RAM, and an hourly cost `E`. On the other hand, when solving the allocation problem on public clouds it is possible to set the number of reserved VMs booked in advance in Listing 7.

C.5 Output Files

When the optimization procedure finishes, the final `Solution` can be retrieved as a JSON file. An example is Listing 8. The file is structured so as to report the overall running cost of the optimal configuration, the time elapsed processing each phase, and detailed information about each class, with the related parameters and results.

In particular, the example result file shows a configuration supporting 20 concurrent users of class R3 by exploiting two 5xlarge VMs, at an overall hourly cost of 1.6758 €/h. As `lstPhases` shows, the optimal solution was obtained in around four hours.

Listing 5: Example JSON file with VM characteristics

```json
{
  "large": {
    "core": 1,
    "memory": 1,
    "provider": "inHouse"
  },
  "medium": {
    "core": 2,
    "memory": 2,
    "provider": "inHouse"
  },
  "xlarge": {
    "core": 4,
    "memory": 4,
    "provider": "inHouse"
  }
}
```

Listing 6: Example JSON file with private cloud parameters

```json
{
  "E": 0.1789,
  "M": 16,
  "N": 7,
  "V": 8
}
```

Listing 7: Example JSON file with public cloud parameters

```json
{
  "R1": {
    "Amazon": {
      "large": {
        "r": 10
      },
      "medium": {
        "r": 20
      },
      "xlarge": {
        "r": 4
      }
    }
  },
  "R2": {
    "Amazon": {
      "large": {
        "r": 16
      },
      "medium": {
        "r": 32
      },
      "xlarge": {
        "r": 8
      }
    }
  }
}
```

Listing 8: Example JSON file with final results

```json
{
  "cost": 1.6758,
  "evaluated": true,
  "id": "3_h20_D1.0E7",
  "lstPhases": [
    {
      "duration": 21154,
      "id": "INIT_SOLUTION"
    },
    {
      "duration": 14430028,
      "id": "OPTIMIZATION"
    }
  ],
  "lstSolutions": [
    {
      "alfa": 7840.0,
      "beta": 392.0,
      "changed": true,
      "cost": 1.6758,
      "deltaBar": 1.3205,
      "duration": 6400256.41025641,
      "error": false,
      "feasible": true,
      "job": {
        "d": 10000000.0,
        "hlow": 14,
        "hup": 20,
        "id": "R3",
        "job_penalty": 28.0,
        "think": 10000.0
      },
      "numCores": 20,
      "numOnDemandVM": 0,
      "numReservedVM": 2,
      "numberContainers": 40,
      "numberUsers": 20,
      "numberVM": 2,
      "parentID": "3_h20_D1.0E7",
      "pos": 0,
      "profile": {
        "cm": 40,
        "cr": 40,
        "mavg": 27034.0,
        "mmax": 79197.0,
        "nm": 288,
        "nr": 302,
        "ravg": 3115.0,
        "rmax": 20522.0,
        "sh1max": 0.0,
        "shtypavg": 13589.0,
        "shtypmax": 89517.0
      },
      "rhoBar": 0.8379,
      "sigmaBar": 0.165,
      "typeVMselected": {
        "id": "5xlarge",
        "r": 24
      }
    }
  ]
}
```

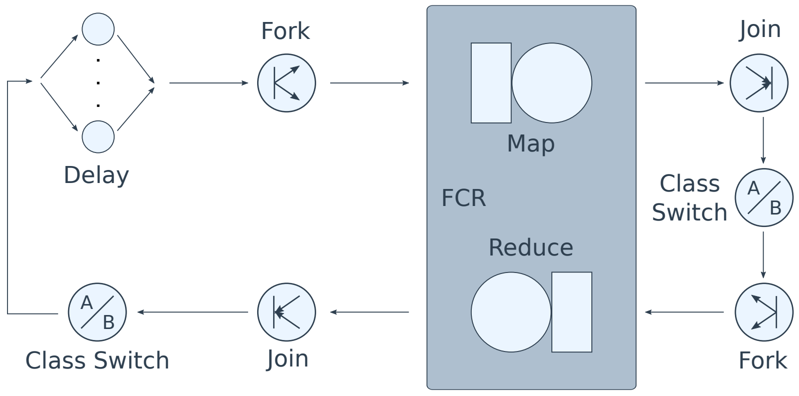


Figure 16: Queueing network model

D Queueing Network Modeling

This appendix discusses the QN model currently embedded into **D-SPACE4Cloud**. After discussing a possible shortcoming of the present formulation, we also describe a recent advancement in JMT that allows for addressing the issue and provide validation results.

D.1 Current queueing network model

In this section we present the QN model we devised and used to obtain an accurate assessment of the job execution times given the cluster configuration. The performance model is depicted in Figure 16. It is a closed QN model where the number of concurrent users gives the overall population and the delay center is characterized by the think time provided via input files, as per Section C.4. When a user submits their job, this is forked into as many map task requests as stated in the relevant job profile, which then enter the finite capacity region (FCR) [3]. FCRs model situations where several service centers access resources belonging to a single limited pool, competing to use them. Hence, the FCR enforces an upper bound on the total number of requests served at the same time within itself, allowing tasks in based on a FIFO queue and supporting prioritization of different classes. The FCR includes two multi-service queues that model the map and reduce execution stages. FCR and multi-service queues capacities are equal to the total number of cores available to the class. In this way, we can model the dynamic assignment of YARN containers to map and reduce tasks whenever they are ready.

Map tasks are executed by the first multi-service queue and synchronize after completion by joining back to a single job request. The reduce phase is modeled analogously, while between the phases there are class switches to enforce that reduce tasks waiting for resources obtain them with priority. Indeed, the YARN Capacity Scheduler implements a FIFO scheduling policy within the same queue and containers are allocated to the next job only when all reduce tasks have obtained enough resources. Reduce tasks have a higher priority than map tasks to enforce this behavior. Note that the map join is external to the FCR in order to model that when map tasks complete they release container cores, which can be assigned to tasks ready in the FCR FIFO queue. Moreover, the reduce fork is also external to the FCR to model correctly applications characterized by a number of reducers larger than the total number of cores. Note that the model in Figure 16 is rather general and can be easily extended to consider also Tez or Spark applications, where a Tez directed acyclic graph node or Spark stage is associated to a corresponding multi-server queue.

Unfortunately, the model presented in Figure 16 has a glitch: it cannot capture the behavior of the *slow start* mechanism, thanks to which reducers within Hadoop 2.x start their shuffle subtask before the map phase has completed to mitigate network congestion. Indeed, such a model enforces a strict separation between the map and reduce phases, thus ruling out the overlapping naturally brought about by the slow start mechanism. Recent advancements in JMT will allow to model more accurately this behavior, as explained in the following.

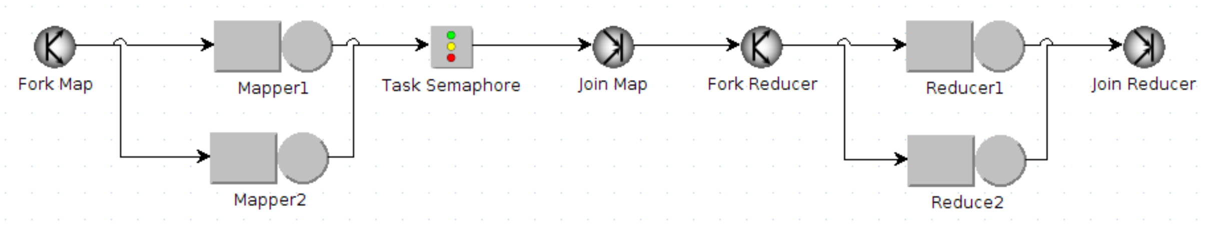


Figure 17: Example MapReduce model with semaphore

Table 6: Measures comparison

Fork Degree	Required Tasks	Expected Task Semaphore RT	Task Semaphore RT	Expected Join RT	Join RT	Expected F/J RT	F/J RT
1	1	0	0	0.5	0.503	1.5	1.5
1	2	0.5	0.5	0	0	1.5	1.498
2	1	0	0	1.25	1.238	2.75	2.746
2	2	0.125	0.125	1.125	1.13	2.75	2.76
2	3	0.5	0.494	0.75	0.754	2.75	2.742
2	4	1.25	1.246	0	0	2.75	2.749

D.2 Task semaphores

When data has to be copied from mapper to reducer nodes, the traffic can be high enough to slow down system performance. In order to address this problem, a mechanism called slow start can be enabled. This mechanism consists in starting the reduce phase while mappers are still processing data. In other words, reducers can start to copy over the output of mappers before the map phase reaches its end. The trigger is a threshold on the percentage of completed map tasks. However, it is important to stress that data merging and the execution of the user-provided reduce function only begins after all mappers finish their processing and the whole reducer inputs are available on the hosting node.

The slow start strategy shows the clear benefit of spreading data copy across a longer time interval, yielding a decrease in network traffic. However, the trade off is that, by starting early, you also assign available containers as reducers early. These resources become unavailable until the map phase is over, which may be a problem if you have many jobs going on at the same time.

In order to describe this behavior into JMT topologies, it was necessary to introduce a new component called *task semaphore*. Its behavior is to stop a percentage of forked tasks in each class until they all reach the semaphore. After the threshold is passed, it lets them pass through altogether, and tasks originated from the same job that arrive later are not blocked. Moreover, this component is only acceptable between a fork and a join because its functionality is based on forked tasks.

Now that JMT has all the necessary components, we can express a model for MapReduce programs configured with slow start. It is shown in Figure 17. The Task Semaphore is introduced after the Mappers (represented by Queues) and before a Join. It represents the data being held on mappers before the copy stage starts. The Join is set with a standard strategy, then it has to wait for all forked tasks to trigger. This represents the reducers waiting for all mappers to start execution.

In the following we report some results to validate the new approach. The response time (RT) of the Task Semaphore is expected to be the same as the Partial Join for the specific number of required tasks, while the Fork/Join RT should not change by the introduction of the new component. To make this comparison an extremely small arriving rate $\lambda = 0.001$ is used, so that the effect of the queue time can be eliminated. The results of this experiment are shown in Table 6.

The second approach used log files, according to Figure 18, in order to verify that the mechanism is working correctly. Both fork degree and number of required tasks were set to 2. The results are reported in Table 7.

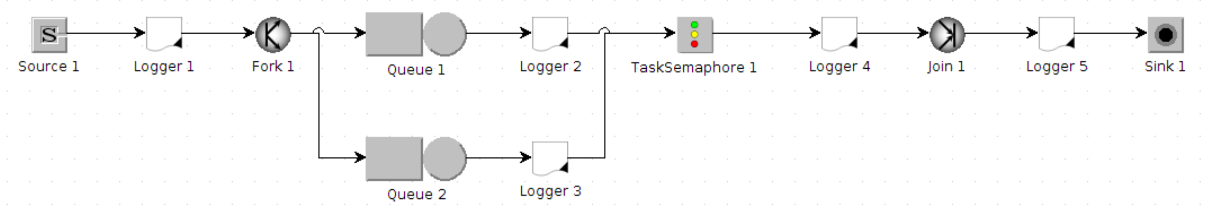


Figure 18: Testing model with loggers

Table 7: Log files results

Logger Name	Timestamp	Job ID
Logger 1	1003.352366854	0
Logger 3	1005.7823473767	2
Logger 2	1006.6586157349	4
Logger 4	1006.6586157349	4
Logger 4	1006.6586157349	4
Logger 3	1006.6786583273	3
Logger 4	1006.6786583273	3
Logger 2	1008.497026407	5
Logger 4	1008.497026407	5
Logger 5	1008.497026407	0

From Tables 6 and 7, we can conclude that the measures are quite close to the expected ones. Moreover, the log files also display a correct algorithm logic. Hence, both analyses validate the desired behavior of the new component.