Co-funded by the European Commission within the Seventh Framework Programme

Project no. 318521

HARNESS

Specific Targeted Research Project
HARDWARE- AND NETWORK-ENHANCED SOFTWARE SYSTEMS FOR CLOUD COMPUTING

Application Characterisation Report
D6.1

Due date: 30 September 2013
Submission date: 30 October 2013

Start date of project: 1 October 2012

Document type: Deliverable
Activity: RTD
Work package: WP6

Editor: Ancuta Iordache (UR1)

Contributing partners: IMP, UR1

Reviewers: Gabriel Figueiredo

Dissemination Level

PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Description
0.1	2013/06/01	Anca Iordache	UR1	Outline
0.2	2013/06/20	Guillaume Pierre	UR1	Document restructuring
0.3	2013/08/28	Anca Iordache	UR1	Added manifest file example
0.4	2013/09/05	Guillaume Pierre	UR1	Updated deliverable according to internal review
0.5	2013/09/17	Guillaume Pierre	UR1	Finalized the deliverable
1.0	2013/10/12	Alexander Wolf	IMP	Final review and edits by Coordinator

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T6.1	Define integrated application performance and resource requirements model	IMP*, EPL, UR1, ZIB, MAX

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

The goal of this deliverable is to describe how to specify the functional and non-functional requirements of applications, based on examples from the *Hardware- and Network-Enhanced Software Systems for Cloud Computing* (HARNESS) validation use cases.

We have designed two domain-specific languages: (i) a *manifest* that allows application developers to express the specifics of an application and the type of resources it needs and (ii) an *service-level objective* (SLO) description language that allows application users to specify the cost and performance they would like to achieve when executing the application. We first describe the two languages and give the manifest file for the AdPredictor validation use case.

Contents

Executive Summary **i**

Acronyms **v**

1 Introduction **1**
1.1 Overview 1
1.2 Requirements 1
1.2.1 Requirements regarding application flexibility 1
1.2.2 Requirements regarding the application description language 2
1.2.3 Requirements regarding SLOs 3
1.3 Report Structure 3

2 Application Model **5**
2.1 Application Organisation 5
2.2 Stakeholders 5
2.3 Application Specification 6

3 Application Manifests **9**
3.1 Example Manifest File 9
3.2 Functional Requirements 11
3.3 Non-Functional Requirements 13

4 Service-Level Objectives **17**
4.1 Types of Supported SLOs 17
4.2 SLO Specification Language 17

5 Example: AdPredictor **19**

6 Conclusions **23**

Acronyms

HARNESS *Hardware- and Network-Enhanced Software Systems for Cloud Computing*. i, 1–3, 5–7, 9, 17, 19, 23

JSON *JavaScript Object Notation*. 7, 8, 23

RTM *reverse time migration*. 3

SLA *service-level agreement*. 2, 3

SLO *service-level objective*. i, 1, 3, 5–7, 11, 17, 23

VM *virtual machine*. 9

1 Introduction

The *Hardware- and Network-Enhanced Software Systems for Cloud Computing* (HARNESS) platform is responsible for the execution of applications on heterogeneous cloud resources, and will provide flexible mechanisms for application specifications to enable dynamic resource allocation that meet given execution deadline and cost objectives. This report outlines general application requirements identified in the context of the HARNESS project. It also presents the methods to specify application characterisation requirements and their purpose in the design of the HARNESS platform.

1.1 Overview

The main purpose of the HARNESS platform is to handle a broad range of applications. This implies that we need to define a general way to describe all these types of applications and to express their requirements with respect to the resources provided by clouds exhibiting a high degree of heterogeneity. For this, we must identify the common characteristics of applications and to study what is the impact of various resource configurations on application performance.

At the core of the HARNESS approach, algorithms receive as input the application requirements and characteristics, the current state of the heterogeneous resources annotated with their performance models, and the provider's internal policies and *service-level objective*s (SLOs). These algorithms output the set of resources that should be assigned to the application. This resource allocation process is not a static process, but rather it should run continuously and adapt to changing conditions, such as when new tasks arrive or resource performance degrades.

The novel aspect of this work is the support of *flexible applications* that may have multiple alternative implementations exhibiting different trade offs between the resources they use and their performance and cost.

1.2 Requirements

This report addresses a number of requirements documented in Deliverables D2.1 and D2.2 [2, 3]. We can classify these requirements into three categories: requirements regarding application flexibility; requirements regarding the application description language; and requirements regarding SLOs.

1.2.1 Requirements regarding application flexibility

Three requirements specify that HARNESS applications may contain multiple alternative implementations. For example, the same algorithm may be implemented in different ways depending on the type of hardware accelerators used for the execution.

R2 (D2.1)	Provide support for flexible applications
The HARNESS platform should support the execution of applications offering a level of freedom in the set of computational, communication, and storage resources they require. This may be realised by a variety of mechanisms: (i) defining rules for horizontal scalability of the application; (ii) authorising application developers to provide multiple implementations of the same functionality featuring different trade offs between the required resources and the achieved performance; (iii) automatically recompiling applications as a means to automate the generation of multiple versions of the application. The platform will autonomously choose one of the available deployment options in order to best enforce the required SLA.	
Task: 6.1, 6.2, 6.3	**Innovation: high Importance: critical** **Dependencies: R3-R4,R6-R8, R11-R12, R14-R16, R20-23**

R26 (D2.2)	The HARNESS platform shall support multiple algorithms
The HARNESS platform shall be able to allow applications to be expressed with multiple choices of algorithm that could be used in their implementation. The platform can then choose among the algorithms to utilise at run time, based on the user's latency or cost target.	
Task: 3.1, 3.2, 6.1, 6.2	**Innovation: high Importance: critical** **Dependencies: R25**

R30 (D2.2)	The HARNESS platform shall support multiple application implementations
The HARNESS platform must be able to support the delta-merge process for different data types that may have differing implementations and vary across a table. As all data types may not be supported across the accelerators, the platform must then choose which implementation to utilise at run time.	
Task: 3.2, 6.3	**Innovation: high Importance: critical** **Dependencies: R1, R2, R9, R11, R12**

1.2.2 Requirements regarding the application description language

Two requirements specify that the HARNESS platform must allow application programmers to describe their application, for example to indicate the type of resources it requires to execute.

R3 (D2.1)	Provide an application description language
The HARNESS platform should provide a description language that application developers can use to specify the functional and non-functional requirements of an application. The functional requirements include the list of software components of the application as well as the location of executables, input and output data. Non-functional requirements include a description of the different ways the application may be deployed on various sets of resources.	
Task: 6.1	**Innovation: medium Importance: critical** **Dependencies: none**

R29 (D2.2)	**The HARNESS platform shall provide support for specification of particular resources**
Since the delta-merge functionality requires close coupling of accelerators as well as specific hardware resources on the accelerators, the HARNESS platform must enable the specification of accelerators based on local memory and hardware requirements.	
Task: 6.1, 6.2	**Innovation: medium Importance: critical Dependencies: R3, R9**

1.2.3 Requirements regarding SLOs

Two requirements specify that users must be able to specify an SLO that indicates the expected level of performance and cost they are ready to accept when executing their applications.

R2 (D2.1)	**Provide an SLA description language**
The HARNESS platform should provide a language to allow application users to specify their expectations of application performance and execution costs.	
Task: 6.1	**Innovation: low Importance: moderate Dependencies: none**

R25 (D2.2)	**The HARNESS platform shall allow the expression of target latency or target cost**
Given the common use case of RTM, the cloud user will likely desire to specify either a cost target (i.e., run in the fastest time not exceeding this cost for the job) or a latency target (i.e., run in minimum cost, taking not longer than this amount of time).	
Task: 6.2, 6.3	**Innovation: medium Importance: moderate Dependencies: None**

1.3 Report Structure

This report is organised as follows. Chapter 2 addresses the application model used by the HARNESS platform with a focus on the description of flexible applications. Chapter 3 presents the description of how HARNESS *application managers* (described in Deliverable D6.3.1 [5]) handle them. Chapter 4 describes the SLOs that HARNESS users may provide when using the platform to run their applications. Chapter 5 shows an example manifest for the AdPredictor validation use case [4]. Finally, Chapter 6 concludes this report and discusses future work.

2 Application Model

The goal of the HARNESS platform is to offer an execution platform for applications making use of heterogeneous cloud resources. Following the requirements mentioned in the previous chapter, the HARNESS platform should be able to support flexible applications capable of using multiple alternative implementations of the same functionality, potentially making use of different types of heterogeneous resources.

This section presents the general requirements on the application description. The platform will support the specification of flexible applications capable of exploiting this heterogeneity and dynamically change its use of resources based on its current workload, the current cost and availability of specialised resources, and the SLO that it is supposed to achieve.

2.1 Application Organisation

To preserve the generality of the platform, flexible applications will be defined as a set of modules that can be deployed over heterogeneous cloud resources. Each module can have one or more independent implementations potentially designed to execute on various types of cloud resources.

From the platform's perspective, each module implementation is an opaque black box defined only by its code, input/output data, and the cloud resources it requires for its execution. In particular, as illustrated in Figure 2.1, the platform has no knowledge about (nor control over) the internal workings of module implementations.

For each implementation in the module, the developer will have to express the functional requirements (runt-time parameters, output) and non-functional requirements (resource types the application supports, constraints related to the number of resources and resource attributes) in a *manifest file*. The manifest language allows one to specify the list of modules and implementations, as well as all the relationships between functional and non-functional application requirements.

2.2 Stakeholders

A platform such as the HARNESS platform involves several types of stakeholders participating in the system in different ways. It is important to distinguish them clearly, as this has an impact on the way applications will be characterised.

- **Platform administrator.** The platform administrator has access to a HARNESS cloud, and offers access to this cloud by providing a HARNESS platform to users. The only task is to set up a generic (application-agnostic) platform together with the HARNESS cloud. The administrator has no inner knowledge about applications that are developed and executed on the platform.

- **Application developer.** Application developers understand the inner working of their application: they know the list of modules that compose an application, the list of implementations of each

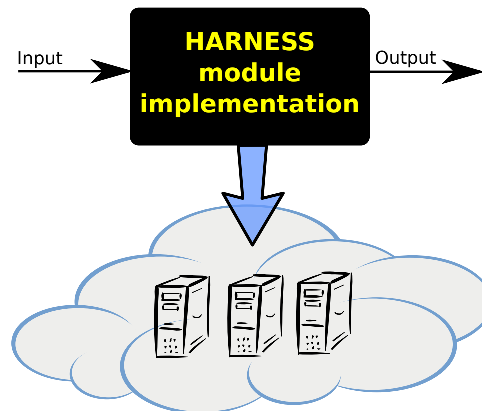


Figure 2.1: A HARNESS module implementation is seen by the platform as a black box executing over a set of cloud resources.

module, the resources each implementation requires for its execution, the relationships between different modules, and so on. To allow the platform to deploy their application, application developers capture their knowledge about the application in the form of an application manifest. We discuss application manifests in detail in Chapter 3.

- **Application user.** Application users know which application they want to execute, which input parameters they want to use, and the performance with which the application should execute. They specify these elements in the form of an SLO, which specifies the particular constraints that are attached to each particular execution of the application. We discuss SLOs in detail in Chapter 4.

Consequently, as illustrated in Figure 2.2, an application execution in the HARNESS platform requires two separate specifications: (i) a specification of the application organisation, captured in the form of an application manifest and written by the application developer, and (ii) a specification of the application parameter values and expected execution performance, captured in the form of an SLO document and written by the application user.

2.3 Application Specification

The HARNESS platform should support the execution of applications offering a level of freedom in the set of computation, communication, and storage devices they require. In order to do this the platform implements several mechanisms:

- allowing application developers to provide several implementations of the same module featuring different trade offs between the required resources and the achieved performance;
- defining rules for horizontal scalability of the application;
- defining the types of resources upon which the application should be executed; and
- allowing application users to specify the objectives of the application execution in terms of cost, performance or a trade off between them.

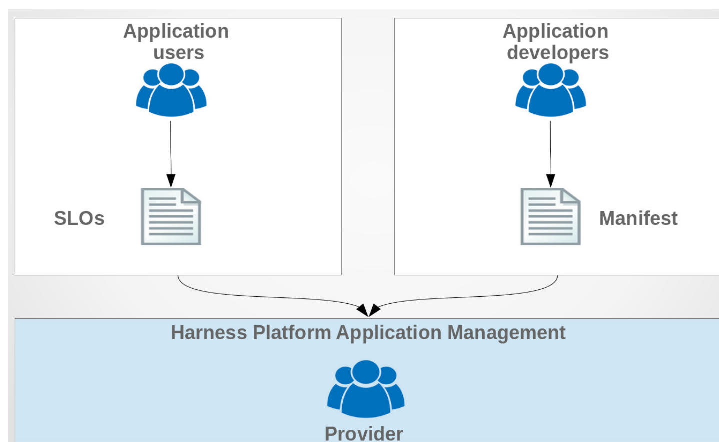


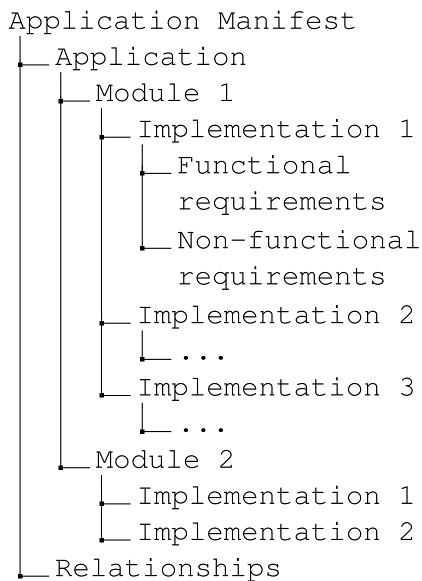
Figure 2.2: A request for executing an application involves two different types of stakeholders.

In order to fulfil the application requirements expressed in deliverables D2.1 [2] and D2.2 [3], the HARNESS platform implements a common interface that supports the description of different types of applications. The common description language splits specifications in two separate types of documents:

- **Application manifest.** The developers of an application must provide the platform with an *application manifest* that specifies the way the application is organised. In particular, the application manifest must contain all the necessary information to let the platform choose an appropriate set of module implementations and resources on which to execute them, deploy the chosen modules, and execute them. The general organisation of an application manifest is depicted in Figure 2.3.

- **Service-level objective.** The user who requests the execution of an application must additionally specify their expectations in terms of execution performance and/or cost. This specification is provided in the form of an *SLO document.*

Application manifests and SLOs are written in *JavaScript Object Notation* (JSON), a standard, open, structured data format widely used by companies such as Yahoo and Google [1]. JSON is supported by many libraries and tools written in many languages, and is the format used for all structured data in ConPaaS [6].



(a) General structure.

```
{
  "Application": [
    {
      "Name": "Module1",
      "Description": "Module description
        ",
      "Implementations": [ {...}, {...}
        ]
    },
    {
      "Name": "Module2",
      "Description": "Module description
        ",
      "Implementations": [ {...}, {...}
        ]
    }
  ],
  "Relationships": [{}]
}
```

(b) JSON representation.

Figure 2.3: Organisation of an application manifest.

3 Application Manifests

As mentioned in Chapter 2, an application consists of a list of modules, each one consisting in turn in a list of implementations. Application developers can specify these functional and non-functional requirements in the form of an *application manifest*.

A HARNESS manifest offers an easy way to provide all the information necessary for the HARNESS platform to efficiently manage application execution. Below, we present a sample of an application manifest that follows the design discussed in Chapter 3 and provides details about application characterisation. We will provide more details in the following sections.

3.1 Example Manifest File

An example manifest file is represented below. It specifies an application composed of one module with a single implementation. This implementation requires two different types of resources: one *virtual machine* (VM) with role "MASTER" and a variable number of VMs with role "SLAVE".

```
{
  "ApplicationName":  "Example application",
  "Author":           "John Doe",
  "PerformanceModel": "/path/to/performance_model",   /* If one was previously generated */
  "Modules" = [
    {
      "ModuleName": "Module1",
      "Implementations": [
        {
          "ImplementationID":     "1",
          "ImplementationName": "First and only implementation",
          "Arguments": [
            {
              "ArgID":        "%arg1",            /* This creates a variable %arg1 that can  */
                                                  /* be referenced elsewhere in the manifest */
              "ArgName":      "Argument name",
              "Type":         "INT",
              "InitValue":    0,                  /* The default value is 0 */
              "Range":        [ 0, 5 ]            /* The platform can choose any value between 0
                  and 5 */
            },
            {
              "ArgID":        "%arg2",
              "ArgName":      "Argument name",
              "Type":         "EXT"               /* The value will be provided by the user in
                 the SLO */
            }
          ],
          "Resources": {
            "Roles": [ "MASTER", "SLAVE" ],
            "Devices": [
              {
                "Role": "MASTER",
```

```
        "Type": "VM",
        "Num": { "Value": "1" },          /* This application needs exactly one master
            */
        "Performance": {
          "RAM": {
            "Value": "%master_ram",       /* The platform will choose a */
            "Range": [ 1, 32 ]            /* value between 1 and 4 GB    */
          }
          "Cores": {
            "Value": "%master_cores"      /* The platform will choose a  */
            "Range": [ 2, 4 ]             /* value between 2 and 4 cores */
          }
        },
        "Addresses": "%master_address"   /* This variable will contain the master IP
            address */
      },
      {
        "Role": "SLAVE",
        "Type": "VM",
        "Num": {
          "Value": "%num_slaves",         /* The platform will choose the        */
          "Range": [ 10, 50 ]             /* number of slaves between 10 and 50 */
        }
        "Performance": {
          "RAM": {
            "Value": "%master_ram",       /* The platform will choose a */
            "Range": [ 8, 32 ]            /* value between 8 and 32 GB   */
          }
          "Cores": {
            "Value": "%master_cores"      /* The platform will choose a  */
            "Range": [ 4, 16 ]            /* value between 4 and 16 cores */
          }
        },
        "Addresses": "%slave_addresses" /* This variable will contain all slave IP
            addresses */
      },
      {
        "Type": "STORAGE",           /* There is no need to define roles here  as      */
                                     /* this application uses only one type of storage */
        "Performance": {
          "Size":       "32",        /* We know we want exactly 32 GB of storage /*
          "IOBw":       "128"        /* We know we want 128 MB/s */
        },
        "Addresses": "%storage_address"
      }
    ],
    "MinBandwidth": "200", /* Minimum available bandwidth in MB/s */
  },
  "GlobalConstraints": [
    "%master_ram > %master_cores",
    "%arg1 = 10 * %num_slaves"
  ],
  "Deployment": {
    "Actions": [
      {
        "ROLE": "MASTER",
        "START": {
          "Script":   "/path/to/master_start_script",
        },
        "STOP": {
          "Script":  "/path/to/master_stop_script",
        }
```

```
              },
              {
                "ROLE": "SLAVE",
                "START": {
                  "Script":   "/path/to/slave_start_script",
                },
                "STOP": {
                  "Script":  "/path/to/slave_stop_script",
                }
              }
            ],
            "EnvironmentVars": {
              "$MASTER_IP": "%MASTER_ADDRESSES", /* $MASTER_IP and $SLAVE_IPS will be exported
                   as     */
              "$SLAVE_IPS": "%SLAVE_ADDRESSES"   /* environment variables in the run-time
                  environment */
            }
          },
          "Activation": {
            "TARGET": {
              "ID": "MASTER"
            },
            "ExecuteCMD": "java /path/to/exe %arg1 %arg2" /* command to start job execution */
          }
        }
      ]
    }
  ]
}
```

3.2 Functional Requirements

Functional requirements in an application manifest describe the list of modules of an application, the way the application should be installed and its software dependencies resolved, and the way to actually start it. The list of dependencies and settings to apply in order to prepare the execution represents the set of static functional requirements. This information is static, and does not depend on decisions taken by the platform.

The most important functional requirements expressed in the manifest are the parameters taken by each module. For simplicity we assume that the order and meaning of the parameters are identical in all the implementations of the same module.

Module parameters may either be specified by the user as part of the application execution request, or chosen automatically by the platform. In this latter case, the platform will first try various values for this parameter during the first runs of the application and study the impact of this parameter's value on application performance. After this initial training phase, the platform will use this knowledge to choose parameter values such that application execution respects the SLO. More details about this process are provided in Deliverable D6.3.1 [5].

Implementation parameters can be specified in the following way in the manifest:

```
...
  "Arguments": [
    {
```

```
      "ArgID":        "%arg1",               /* This creates a variable %arg1 that can
          */
                                             /* be referenced elsewhere in the manifest
                                                 */
      "ArgName":      "Argument name",
      "Type":         "INT",
      "InitValue":    0,                     /* The default value is 0 */
      "Range":        [ 0, 5 ]               /* The platform can choose any value
          between 0 and 5 */
    },
    {
      "ArgID":        "%arg2",
      "ArgName":      "Argument name",
      "Type":         "EXT"                  /* The value will be provided by the user
          in the SLO */
    }
  ],
...
```

As we can see, for each parameter, the developer can specify:

- a parameter identifier, used when expressing dependencies on the resources;
- an initial value of the parameter, representing the starting point for parameter space exploration in the profiling process; and
- a range or list of values, used in the profiling process to explore the parameter-dependent behaviour of the implementation.

The static requirements can be expressed the following way:

```
...
          "Deployment": {
            "Actions": [
              {
                "ROLE": "MASTER",
                "START": {
                  "Script":    "/path/to/master_start_script",
                },
                "STOP": {
                  "Script":   "/path/to/master_stop_script",
                }
              },
              {
                "ROLE": "SLAVE",
                "START": {
                  "Script":    "/path/to/slave_start_script",
                },
                "STOP": {
                  "Script":   "/path/to/slave_stop_script",
                }
              }
```

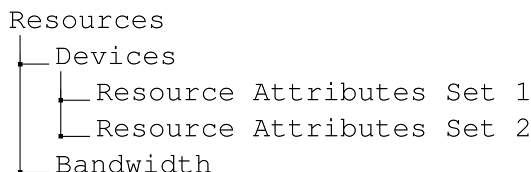
```
            ],
            "EnvironmentVars": {
              "$MASTER_IP": "%MASTER_ADDRESSES", /* $MASTER_IP and $SLAVE_IPS will
                  be exported as      */
              "$SLAVE_IPS": "%SLAVE_ADDRESSES"    /* environment variables in the
                  run-time environment */
            }
          },
          "Activation": {
            "TARGET": {
              "ID": "MASTER"
            },
            "ExecuteCMD": "java /path/to/exe %arg1 %arg2" /* command to start job
                execution */
          }
...
```

As we can observe, there are two main operations to be performed that require static information:

- **Deployment.** Installing the software stack and preparing the execution environment. This requires preparing the execution environment as well as defining settings such that multiple modules can later communicate with each other. Such bindings are established thanks to the `GlobalVars` field, which stores information exposed to each process.
- **Activation.** Execution information specifying the end point where to submit the application and how to launch its execution.

3.3 Non-Functional Requirements

Non-functional requirements describe the set of resources that are necessary to execute the application. As shown below, each application can require multiple types of resources, possibly with bandwidth requirements between these resources.



As for the functional requirements, application developers can either precisely specify the resources needed by their application, or give some leeway for the platform to automatically tune certain parameters.

An example of resource requirements is provided below:

```
...
    "Resources": {
      "Roles": [ "MASTER", "SLAVE" ],
      "Devices": [
        {
```

```
    "Role": "MASTER",
    "Type": "VM",
    "Num": { "Value": "1" },        /* This application needs exactly one
        master */
    "Performance": {
      "RAM": {
        "Value": "%master_ram",     /* The platform will choose a */
        "Range": [ 1, 32 ]          /* value between 1 and 4 GB   */
      }
      "Cores": {
        "Value": "%master_cores"    /* The platform will choose a  */
        "Range": [ 2, 4 ]           /* value between 2 and 4 cores */
      }
    },
    "Addresses": "%master_address"  /* This variable will contain the master
        IP address */
  },
  {
    "Role": "SLAVE",
    "Type": "VM",
    "Num": {
      "Value": "%num_slaves",       /* The platform will choose the        */
      "Range": [ 10, 50 ]           /* number of slaves between 10 and 50 */
    }
    "Performance": {
      "RAM": {
        "Value": "%master_ram",     /* The platform will choose a */
        "Range": [ 8, 32 ]          /* value between 8 and 32 GB  */
      }
      "Cores": {
        "Value": "%master_cores"    /* The platform will choose a  */
        "Range": [ 4, 16 ]          /* value between 4 and 16 cores */
      }
    },
    "Addresses": "%slave_addresses" /* This variable will contain all slave
        IP addresses */
  },
  {
    "Type": "STORAGE",          /* There is no need to define roles here  as
          */
                                /* this application uses only one type of
                                   storage */
    "Performance": {
      "Size":      "32",        /* We know we want exactly 32 GB of storage /*
      "IOBw":      "128"        /* We know we want 128 MB/s */
    },
    "Addresses": "%storage_address"
  }
 ],
 "MinBandwidth": "200", /* Minimum available bandwidth in MB/s */
}
...
```

The fields starting with `%` are variables and may be dependent on other fields. This is why values are going to be assigned to them at run time after the constraints are processed.

Application developers can also reduce the search space by giving constraints between multiple parameters that the platform should optimise. Here is an example of such constraints:

```
...
   "GlobalConstraints": [
     "%master_ram >  %master_cores",
     "%arg1 = 10 * %num_slaves"
   ]
...
```

4 Service-Level Objectives

SLO descriptions are the way by which HARNESS users can express their wishes regarding the performance/cost trade offs that they are willing to accept when executing an application. Additionally, the SLO also specifies the functional requirements of an application execution request, such as the location of input data and the value of configuration parameters.

4.1 Types of Supported SLOs

As specified in Requirement R25 [3], SLOs can be expressed in two different ways:

- The user specifies an expected level of performance. In this case, the platform will try to minimise costs while respecting the performance target.
- The user specifies an expected execution cost. In this case the platform will try to maximise performance without exceeding the allocated budget.

A third option would consist of letting the user define a cost function that encapsulate the exact performance/cost trade offs that are considered acceptable. Implementing such SLOs would not be difficult, but experience shows that users find such cost functions confusing and hard to define. As a consequence, we will initially focus on simple SLOs that define either a target performance or a target execution cost.

4.2 SLO Specification Language

An example SLO is shown below:

```
{
  "SLO": {
    "ManifestUrl": "path/to/application/manifest"
    "ExecutionArgs": [
      {"ArgID": "1", "Value": "500"},
      {"ArgID": "2", "Value": "200"}
    ],
    "Objective": {
      Constraints: [ "%budget <= 100" ]  /* spend no more than $100 */
      Optimization: "%execution_time"    /* minimize latency while respecting the
          budget */
    }
  }
}
```

The SLO first contains a link to the application manifest to which it refers. It then contains the parameter values that this execution should use, and a constraint regarding the maximum budget that the user is

ready to spend. The optimisation objective consists of minimising the execution time without exceeding the budget.

5 Example: AdPredictor

This chapter presents an example of a manifest file for one of the three HARNESS validation use cases, AdPredictor. The manifest file describes AdPredictor as an application consisting of one module with two different implementations: a sequential and a MapReduce implementation. Both implementations process synthetic data generated by the deployment scripts. The arguments to all implementations must have the same order and significance.

The sequential implementation requires one machine. AdPredictor is installed in it through the script specified in the `Deployment` field `START`. No stop application script is required as it does not support scaling. This implementation processes synthetic data (generated by the start script) whose location is passed by the the first argument.

The AdPredictor's MapReduce implementation runs on a cluster of machines with a master/slave architecture. The manifest file contains constrains specifying the requirement of one master machine and at least one slave machine. The MapReduce processes and all other dependencies are installed through the start scripts. Having a more complex configuration (machines have different roles), the manifest provides scripts (start and stop processes) for handling each type of machine in the system.

For each implementation, there are specified environment variables that will be exported to all the machines from the configuration. Through this, we solve inter-resource dependencies.

```
{
  "ApplicationName":  "AdPredictor",
  "Author":           "HarnessUser",
  "Modules" = [
    {
      "ModuleName": "AdPredictor",
      "Implementations": [
      /**************************************************/
      /** Description of the sequential implementation **/
      /**************************************************/
        {
          "ImplementationID":   "1",
          "ImplementationName": "Sequential Implementation",
          "Arguments": [
            {
              "ArgID":        "%arg1",
              "ArgName":      "Input data path",
              "Type":         "STR",
              "InitValue":    "sample-data/I100F3V3.txt",
              "Values":       [ "sample-data/I100F3V3.txt" ]
            },
            {
              "ArgID":        "%arg2",
              "ArgName":      "Output data path",
              "Type":         "STR",
              "InitValue":    "/tmp/output",
              "Values":       [ "/tmp/output" ]
            }
          ],
```

```
      "Resources": {
        "Roles": [ "MACHINE"],
        "Devices": [
          {
            "Role": "MACHINE",
            "Type": "VM",
            "Num": { "Value": "1" },          /* This application needs only one machine */
            "Performance": {
              "RAM": {
                "Value": "%machine_ram",      /* The platform can choose a */
                "Range": [ 1024, 4096 ]           /* value between 1 and 4 GB    */
              }
              "Cores": {
                "Value": "%machine_cores"     /* The platform can choose a  */
                "Range": [ 2, 8 ]             /* value between 2 and 8 cores */
              }
            },
            "Addresses": "%machine_address"
          }
        ]
      },
      "GlobalConstraints": [],
      "Deployment": {
        "Actions": [
          {
            "ROLE": "MACHINE",
            "START": {

            /* This script is installing the application on the machine */
            /* and generates the input data to process.
              "Script":  "http://public.rennes.grid5000.fr/~aiordache/harness/apps/
                  adpredictor/adpredictor_install_script.sh",

            }
          }
        ],
        "EnvironmentVars": {
           "$INPUT_DIR"   : "%arg1",
           "$OUTPUT_DIR"  : "%arg2"
        }
      },
      "Activation": {
        "TARGET": {
          "ID"     : "MACHINE",
          "USER"  : "root"
        },
        "ExecuteCMD": "java uk.ac.imperial.adpredictor.AdPredictor %arg1 %arg2"
      }
    },

  /****************************************************/
  /** Description of the MapReduce implementation   **/
  /****************************************************/

    {
      "ImplementationID": "2",
      "ImplementationName": "MapReduce Implementation",
      "Arguments": [
        {
          "ArgID": "%arg1",
          "ArgName": "Input data path",
          "Type": "STR",
```

```
        "InitValue": "/user/hadoop/input",
        "Values": [ "/user/hadoop/input" ]
      },
      {
        "ArgID": "%arg2",
        "ArgName": "Output data path",
        "Type": "STR",
        "InitValue": "/user/hadoop/output",
        "Values": [ "/user/hadoop/output" ]
      }
    ],
    "Resources": {
      "Roles": [ "MASTER", "SLAVE" ],
      "Devices": [
        {
          "Role": "MASTER",
          "Type": "VM",
          "Num": {
            "Value": 1
          },
          "Performance": {
            "RAM": {
              "Value": "%master_ram",
              "Range": [ 1024, 4096 ]
            },
            "Cores": {
              "Value": "%master_cores",
              "Range": [ 2, 4 ]
            }
          },
          "Addresses": "%master_addresses"
        },
        {
          "Role": "SLAVE",
          "Type": "VM",
          "Num": {
            "Value": "%slave_num"
          },
          "Performance": {
            "RAM": {
              "Value": "%slave_ram",
              "Range": [ 2048, 4096 ]
            },
            "Cores": {
              "Value": "%slave_cores",
              "Range": [ 2, 8 ]
            }
          },
          "Addresses": "%slave_addresses"
        }
      ],
      "MinBandwidth": "200"
    },
    "GlobalConstraints": ["%slave_num >= 1"],          /* requires at least one slave
        machine */
      "Deployment": {
        "Actions": [
          {
            "ROLE": "MASTER",
            "START": {
              "Script": "http://public.rennes.grid5000.fr/~aiordache/harness/apps/
                  adpredictor/start_hadoop_master_processes_and_install_adpredictor.sh"
```

```
                  },
                  "STOP": {
                    "Script": "http://public.rennes.grid5000.fr/~aiordache/harness/apps/
                        adpredictor/stop_hadoop_master_processes.sh"
                  }
                },
                {
                  "ROLE": "SLAVE",
                  "START": {
                    "Script": "http://public.rennes.grid5000.fr/~aiordache/harness/apps/
                        adpredictor/start_hadoop_slave_processes.sh"
                  },
                  "STOP": {
                    "Script": "http://public.rennes.grid5000.fr/~aiordache/harness/apps/
                        adpredictor/stop_hadoop_slave_processes.sh"
                  }
                }
              ],
              "EnvironmentVars": {
                "$MASTER_IP": "%master_address",
                "$INPUT_DIR": "%arg1",
                "$OUTPUT_DIR": "%arg2"
              }
            },
            "Activation": {
              "TARGET": {
                "ID": "MASTER",
                "USER": "hadoop"
              },
              "ExecuteCMD": "hadoop jar MRAdPredictor.jar MRAdPredictor %arg1 %arg2"
            }
          }
        ]
      }
    ]
}
```

6 Conclusions

This report describes a specification language for application manifests and SLOs in the HARNESS platform. These languages allow developers to easily describe the specifics of their applications, including multiple alternative implementations and their respective needs in terms of cloud resources. Similarly, cloud users can easily request application executions while specifying their expectations of performance and execution cost. Both types of documents are encoded in JSON, which makes it trivially easy to generate and parse in any programming language.

Although these two languages may slightly evolve in the future, if the need arises, we plan to keep such specifications as simple as possible for application developers and cloud users.

Bibliography

[1] D. Crockford. The application/json media type for Javascript object notation (JSON). RFC 4627, July 2006.

[2] FP7 HARNESS Consortium. General requirements. Project Deliverable D2.1, 2013.

[3] FP7 HARNESS Consortium. Industrial requirements. Project Deliverable D2.2, 2013.

[4] FP7 HARNESS Consortium. Validation plan. Project Deliverable D2.3, 2013.

[5] FP7 HARNESS Consortium. Heterogeneous platform implementation (initial). Project Deliverable D6.3.1, 2013.

[6] G. Pierre and C. Stratan. ConPaaS: A platform for hosting elastic cloud applications. *IEEE Internet Computing*, 16(5):88–92, Sept. 2012.