Validation and Result Analysis

Project Deliverable D7.4.2, revision 3

Craig Sheridan, Darren Whigham, Claire Stewart (FLEX), Jörg Domaschka, Athanasios Tsitsipas, Christopher Hauser (UULM), Zafeirios Papazachos (QUB), Ahmed Ali-Eldin, Jakub Krzywda (UMU), James Byrne (DCU)

Due date: 30 September 2016
Delivery date: 02 February 2017

This project is funded by the European Union under grant agreement no. 610711
## Dissemination Level

<table>
<thead>
<tr>
<th>X</th>
<th>PU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PU</td>
<td>Public</td>
</tr>
<tr>
<td>PP</td>
<td></td>
<td>Restricted to other programme participants (including the Commission Services)</td>
</tr>
<tr>
<td>RE</td>
<td></td>
<td>Restricted to a group specified by the consortium (including the Commission Services)</td>
</tr>
<tr>
<td>CO</td>
<td></td>
<td>Confidential, only for members of the consortium (including the Commission Services)</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Change</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0.1</td>
<td>10/08/16</td>
<td>Document started with structure placed</td>
</tr>
<tr>
<td>0.2</td>
<td>10/08/16</td>
<td>Adjust the validation for the scientific computing application</td>
</tr>
<tr>
<td>0.3</td>
<td>02/09/16</td>
<td>Input from QUB</td>
</tr>
<tr>
<td>0.4</td>
<td>10/09/16</td>
<td>Input from UULM</td>
</tr>
<tr>
<td>0.4.1</td>
<td>11/09/16</td>
<td>Description of OpenStack VMI</td>
</tr>
<tr>
<td>0.5</td>
<td>19/09/16</td>
<td>Update from all partners</td>
</tr>
<tr>
<td>0.6</td>
<td>22/09/16</td>
<td>First Review</td>
</tr>
<tr>
<td>0.7</td>
<td>28/09/16</td>
<td>Addressing comments for UULM</td>
</tr>
<tr>
<td>0.8</td>
<td>28/09/16</td>
<td>Address comments for UMEA</td>
</tr>
<tr>
<td>0.9</td>
<td>28/09/16</td>
<td>Address comments for QUB</td>
</tr>
<tr>
<td>0.91</td>
<td>29/09/16</td>
<td>References to D6.5</td>
</tr>
<tr>
<td>0.92</td>
<td>29/09/16</td>
<td>Combine input from all partners</td>
</tr>
<tr>
<td>1.0</td>
<td>30/09/16</td>
<td>Final Proofreading</td>
</tr>
<tr>
<td>1.1</td>
<td>04/11/16</td>
<td>Added Dataplay Validation case</td>
</tr>
<tr>
<td>1.2</td>
<td>05/11/16</td>
<td>Finalised Dataplay validation, fixed license</td>
</tr>
<tr>
<td>2.0</td>
<td>06/11/16</td>
<td>Release</td>
</tr>
<tr>
<td>2.1</td>
<td>30/11/16</td>
<td>Add description of OpenCast Application</td>
</tr>
<tr>
<td>2.2</td>
<td>12/12/16</td>
<td>Add description of use case and black/grey/white box models</td>
</tr>
<tr>
<td>2.3</td>
<td>28/01/17</td>
<td>Add Opencast white box model</td>
</tr>
<tr>
<td>2.4</td>
<td>30/01/17</td>
<td>Added section on opencast performance modelling</td>
</tr>
<tr>
<td>2.5</td>
<td>30/01/17</td>
<td>Added section on worker sizing</td>
</tr>
<tr>
<td>2.6</td>
<td>31/01/17</td>
<td>Added section on evaluation scenario</td>
</tr>
<tr>
<td>2.7</td>
<td>31/01/17</td>
<td>Incorporated results and evaluation summary</td>
</tr>
<tr>
<td>3.0</td>
<td>02/02/17</td>
<td>Final polishing of the sections</td>
</tr>
</tbody>
</table>
This document describes the validation of the CACTOS software components. This validation has been completed in two iterations. The first iteration aimed to deliver a current validation of the CACTOS Runtime tools as they were available at that time. The second iteration contained in this document presents the validation of the final iteration of the CACTOS tooling. These tools are validated within multiple use case scenarios. For each scenario in CACTOS (Business Analytics, Scientific Computing and Cloud Application) a set of validation “scenarios” have been discussed and detailed to the point where they can be applied to the concrete technical solution and deployment. By the end of the project scenarios from all stakeholders detailed within this document have either been fulfilled or partly fulfilled.

The validation results have been defined in two iterations:

- First iteration is reported in deliverable D7.4.1 at PM 18 and was used to derive future actions.
- Second iteration is reported in this deliverable D7.4.2 at PM 36, with respect to future commercialisation.

The document consists of five chapters out of which three cover the validation process: Defining the goals that the software should meet, validation of individual tools and validation by use case. What is new in this document compared to its predecessor D7.4.1 and D7.3.2 is that validation scenarios have also been defined for the virtual middleware integration components.

The document ends by bringing together all results for a final conclusion.
# TABLE OF CONTENTS

EXECUTIVE SUMMARY .................................................. I

TABLE OF CONTENTS .................................................... II

LIST OF FIGURES ........................................................ V

LIST OF TABLES .......................................................... VI

ABBREVIATIONS .......................................................... VII

I. INTRODUCTION ......................................................... 1

II. UPDATED VALIDATION GOALS ...................................... 2

III. VALIDATION BY TOOLS .............................................. 3

1. CactoSim ................................................................. 3

2. CactoOpt ................................................................. 3

   a) Scenarios and Functional Requirements .................................. 3
   b) Criteria ........................................................................ 6
   c) Future Commercialisation and Features ...................................... 7

3. CactoSacle ................................................................. 7

   a) Description .................................................................. 7
   b) Steps ......................................................................... 7
   c) Criteria ....................................................................... 8
   d) Results ....................................................................... 9
   e) Future Commercialisation and Features .................................... 9

4. Virtual Middleware Integration ....................................... 9

   FCO ........................................................................... 10

   a) Description ................................................................ 10
   b) Steps ....................................................................... 10
   c) Criteria ..................................................................... 11
   d) Results ..................................................................... 11
   e) Future Commercialisation and Features ................................. 11

   OPENSTACK .......................................................... 11

   f) Description ................................................................ 11

   g) VMI.OS1 Deploying and Deleting a Virtual Machine .............. 17
   h) VMI.OS2 Migrating a Virtual Machine ................................. 18
   i) VMI.OS3 Switching On/Off a Physical Machine .................. 19
   j) VMI.OS4 Resource-Control on Hypervisor Level .................. 19


IV. VALIDATION BY USE CASE SCENARIOS

1. FLEXIANT BUSINESS ANALYTICS
   a) OVERVIEW 22
   b) NODE LOAD DISTRIBUTION 22
   c) NODE FAULT TOLERANCE 24
   d) OVERALL RESULTS AND FUTURE ACTIONS 25

2. UULM SCIENTIFIC COMPUTING
   a) VALIDATION OF DEPLOYMENT 26
   b) SC.1 RUNNING MOLPRO IN A VIRTUALISED ENVIRONMENT 27
   c) SC.2 DEPLOYING MOLPRO THROUGH CACTOS 27
   d) SC.3 MONITORING MOLPRO INSTANCES 28
   e) SC.4 MINE MOLPRO TRACES 30
   f) SC.5 PREDICTION OF EXECUTION TIME AND EXECUTION PHASES 30
   g) SC.6 ONLINE PHASE DETECTION OF RUNNING APPLICATIONS 32
   h) SC.7 FAILURE DETECTION AND SNAPSHOTS 32
   i) SC.8 APPLICATION RESTART FROM SNAPSHOT IN CASE OF FAILURES 33
   j) SC.9 CONCURRENT USAGE OF PHYSICAL RESOURCES 33
   k) SC.10 PHASE-AWARE SCHEDULING 35
   l) SC.11 DETECT LACK OF RESOURCES 36
   m) SC.12 SIMULATE OF SPARE RESOURCES 37
   n) SC.13 SIMULATE CHANGE OF RESOURCE ALLOCATION 38
   o) SC.14 CONTROL POWER STATE OF PHYSICAL RESOURCES 38
   p) OVERALL RESULTS AND FUTURE ACTIONS 39

3. DATAPLAY CLOUD APPLICATION
   a) CHANGES SINCE D7.3.2 41
   b) CA.1 INITIAL SYSTEM SETUP 41
   c) CA.2 USER-DRIVEN OPTIMISATION 42
   d) CA.3 AUTO-SCALING 43
   e) CA.4 DISTRIBUTING THE DATAPLAY APPLICATION ACROSS MULTIPLE SITES 44
   f) CA.5 APPLICATION LEVEL LOAD-BALANCING ALGORITHM EFFECTS ON OVERLOADED APPLICATIONS 46

   g) OVERALL RESULTS AND FUTURE ACTIONS 46

V. TUTORIAL: ADDING A NEW APPLICATION TO CACTOS

1. THE OPENCAST APPLICATION
   a) APPLICATION OVERVIEW 48
   b) USE CASE DESCRIPTION: OPTIMISED ENCODING 49
2. CACTOS SUPPORT FOR OPENCAST
   a) CACTOS APPLICATION MODELS 50
   b) STEP 1: CREATION OF AN OPENCAST WHITEBOX MODEL TEMPLATE 53
   c) STEP 2: CREATION OF OPENCAST DEPLOYMENT SCRIPTS 54
   d) STEP 3: OPENCAST PERFORMANCE MODELLING 55
   e) STEP 4: OPENCAST-SPECIFIC SENSOR 58
   f) STEP 5: EVALUATION AND SIMULATION 58

3. SUMMARY AND GENERAL RECOMMENDATIONS 61
4. SCREENCAST AND INSTALLATION GUIDELINES 63

VI. CONCLUSIONS AND FUTURE ACTIONS 64

REFERENCES 65
LIST OF FIGURES

Figure 1 Validation Scenario 1 for CactoOpt ................................................................. 4
Figure 2 Validation Scenario 3 for CactoOpt ................................................................. 5
Figure 3 Validation Scenario 4 for CactoOpt ................................................................. 6
Figure 4 Screenshot of the CACTOS Dashboard ............................................................ 12
Figure 5 VMI OpenStack ............................................................................................... 12
Figure 6 Interception of OpenStack Command when Creating a Virtual Machine .......... 15
Figure 7 Snapshot of CACTOS GUI that triggers a Dataplay Deployment ...................... 42
Figure 8 Development of network load during a Dataplay run ........................................ 45
Figure 9 Development of Masters and Requests for Dataplay Deployment 196608 .............. 45
Figure 10 Development of power consumption and # virtual machines during that time interval ......................................................................................................................... 45
Figure 11 User Screen for Creating a New Recording Event in Opencast (Step 1: Metadata) ......................................................................................................................... 50
Figure 12 Screenshot of OpenStack Dashboard of a Virtual Machine with CACTOS-injected meta-data. The image shows a Virtual Machine running the Molpro application. The meta-data lists the application name (Molpro), the name of the (only) application component (molproComponent), as well as several Molpro-specific parameters that can be used to identify similar runs of the same application (molpro-input : “lccsd-sp01", molproType : “molpro-lccsd", molpro-size : 512). ................................................................. 51
Figure 13 Screenshot from Eclipse of Opencast White-box Model Template .................... 53
Figure 14 Opencast Deployment Dependencies ................................................................ 54
Figure 15 User Screen for Creating a New Recording Event in Opencast (Step 3: Selection of Workflow) ........................................................................................................ 55
Figure 16 Sensor Configuration for Opencast’s Admin Component ................................. 58
Figure 17 Evaluation Workload, Simulated Scaling, and Real-world Scaling .................... 59
Figure 18 Example Script for Issuing Load Towards an Opencast Installation .................. 60
LIST OF TABLES

TABLE 1: VALIDATION GOALS FROM D7.3.1 ........................................................................................................2
TABLE 2: FCO VMI ACTIONS .......................................................................................................................... 10
TABLE 3 OPTIMISATION ACTIONS SUPPORTED BY OpenStack VMI ............................................................... 16
TABLE 4 OVERVIEW OF OpenStack VALIDATION RESULTS ......................................................................... 39
TABLE 5 DATAPLAY CONFIGURATIONS AS USED IN CELAR ........................................................................ 42
TABLE 6 OVERVIEW OF ENTERPRISE APPLICATION VALIDATION RESULTS ................................................. 46
TABLE 7 OPENCAST COMPONENT OVERVIEW ............................................................................................... 49
TABLE 8 OVERVIEW OF DEFAULT OPENCAST WORKFLOW STEPS WITH DURATION AND MAXIMUM MEASURED CPU DEMAND FOR A TYPICAL 90 MINUTE LECTURE RECORDING AND A 4-CORE WORKER WITH 8 GB OF MEMORY (RECORDING INCORPORATES ONLY THE RECORDING OF THE SLIDES AS SHOWN ON THE BEAMER SCREEN) .................................................................................................................. 56
TABLE 9 EXECUTION TIME PROCESSING A SINGLE 90 MINUTE VIDEO DEPENDING ON THE NUMBER OF WORKERS .................................................................................................................................................. 57
TABLE 10 TOTAL EXECUTION TIME OF 1-4 PARALLEL JOBS ON ONE WORKER WITH 4 CORES ...................... 57
TABLE 11 COMPARISON OF DIFFERENT OPENCAST PROPERTIES RELATED TO OPERATION .................. 61
TABLE 12 TIME NEEDED MAKING OPENCAST CACTOS-ENABLED ..................................................................... 62
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACTOS</td>
<td>Context-Aware Cloud Topology Optimisation and Simulation</td>
</tr>
<tr>
<td>FCO</td>
<td>Flexiant Cloud Orchestrator</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VMI</td>
<td>Virtual Middleware Integration</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>BC</td>
<td>Business Use Case</td>
</tr>
<tr>
<td>SC</td>
<td>Scientific Use Case</td>
</tr>
<tr>
<td>DFT</td>
<td>Density Functional Theory</td>
</tr>
<tr>
<td>LCCSD</td>
<td>(Linearized) Coupled Cluster With Single And Double Excitation</td>
</tr>
<tr>
<td>SSD</td>
<td>Solid State Disk</td>
</tr>
<tr>
<td>HDD</td>
<td>Hard Disk Drive</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>IPMI</td>
<td>Intelligent Platform Management Interface</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

This document provides final results from the validation of the CACTOS Runtime Toolkit v2 as presented in D5.2.2. This report describes work done towards a validation of the CACTOS solution with validation of individual tools and use case scenarios. Each individual tool will be validated before individual sections of the Business, Scientific and Cloud Application scenario are validated. In order to perform the validation, we use the validation scenarios defined in D7.3.1 and D7.3.2. This document contains the results of each use case scenario, done by the corresponding partners.

The CACTOS Runtime Toolkit consists of three main components: CactoScale collects information about the infrastructure such as the physical hardware, the virtualised interface such as running virtual machines and collects load information from the running system covering both physical and virtual nodes and networks. CactoOpt is concerned with resource allocation and application placement. In particular, it selects the best fitting resources for application deployment taking workload analyses from CactoScale and application models into account. Finally, the Virtual Middleware Integration (VMI) is used to perform optimisation actions within the target cloud platform.

CACTOS has defined three scenarios from different application domains that shall be used to validate the CACTOS software and prove its applicability and usefulness. The business analytics use cases targets the monitoring of data centres used by cloud providers. This is used for the best fitting hardware resources as well as the detection of hardware failures and malfunctioning and the automatic triggering responses. The scientific computing use cases deals with the execution of chemical computations on virtualised compute resources. This requires sophisticated placement and controlling of virtual machines, the application’s input parameters, and overall system load. The Cloud Application use case represents scenarios for any modern, elastic, multi-tier web application.

This deliverable covers the final results from the validation of the CACTOS Runtime Toolkit and in addition provides the final results from the use case scenarios. It includes the validation of the CACTOS Runtime Toolkit.
II. UPDATED VALIDATION GOALS

Validation goals are set on the functionalities or capabilities that the software provides. The capabilities of the software product are predefined as stated within deliverable D7.3.1. It needs to be proven if the software behaves as defined. The CACTOS Toolkit consists of three main components – CactoScale, CactoOpt and CactoSim – each validation goal triggers functionalities in one or more of these components, therefore Table 1 maps the goal to the affected components.

Table 1: Validation Goals from D7.3.1

<table>
<thead>
<tr>
<th>#</th>
<th>GOAL</th>
<th>DESCRIPTION OF GOAL</th>
<th>AFFECTED COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Resource Provisioning</td>
<td>Capability to provide and manage resources in a</td>
<td>CactoScale,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>heterogeneous infrastructure</td>
<td>CactoOpt</td>
</tr>
<tr>
<td>2</td>
<td>Power and Cost Optimization</td>
<td>Ability to change resources of a virtual system to</td>
<td>CactoScale,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reduce power consumption</td>
<td>CactoOpt</td>
</tr>
<tr>
<td>3</td>
<td>Heterogeneous Data Centre</td>
<td>Compatibility to manage resources in non-homogeneous</td>
<td>CactoScale,</td>
</tr>
<tr>
<td></td>
<td>Compatibility</td>
<td>data centres</td>
<td>CactoOpt</td>
</tr>
<tr>
<td>4</td>
<td>Gather Characteristics</td>
<td>Get the status of a data centre system</td>
<td>CactoScale</td>
</tr>
<tr>
<td>5</td>
<td>Usage Prediction</td>
<td>Allow the usage prediction for the future based on</td>
<td>CactoScale,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>models or current usage</td>
<td>CactoSim</td>
</tr>
<tr>
<td>6</td>
<td>Event Driven Optimization</td>
<td>Based on the preferences the toolkit automatically or</td>
<td>CactoScale,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interactively attaches or detaches resources</td>
<td>CactoOpt</td>
</tr>
<tr>
<td>7</td>
<td>Resource Conflict Prediction</td>
<td>Warn before a free resource runs short (like storage</td>
<td>CactoScale,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or free CPU cores)</td>
<td>CactoSim</td>
</tr>
</tbody>
</table>
III. VALIDATION BY TOOLS

This section describes the individual tools within the CACTOS Toolkit and details the steps and requirements used to validate each individual tool. Each individual tool will be described as well as the steps; criteria and results being detailed.

1. CactoSim

Cloud computing data centres are complex systems with a high degree of heterogeneity and a large number of different elements (e.g. racks, nodes, hard disk drives, virtual machines) with various forms of interactions and dependencies. Any system with these characteristics is exceptionally difficult to manage as any decision to make a change or react to an exception can have significant operational and cost implications. To support decisions of this nature, the inherent relationships within the system must be understood and incorporated into an evaluation process. The use of CactoSim gives the ability to model system behaviour and produce information for large-scale data centres without starting a single VM. In order to cover validation goals, the CACTOS deliverable “D6.5 Final Results from Optimisation Models Validation and Experimentation” delivered at the same time as this deliverable provides the reader with evaluation scenarios across the three CACTOS use cases as well as evaluation of the CACTOS Prediction Toolkit across each of these scenarios. Please refer to D6.5 for this validation.

2. CactoOpt

CactoOpt is the CACTOS infrastructure optimisation tool. CactoOpt recommends actions to optimise the operation of a data centre towards an optimization goal. Valid optimisation goals supported by CactoOpt include load balancing the datacentre to increase the overall performance of the applications, consolidation to increase the utilization of the running servers and reduce the running costs by shutting down unnecessary machines, energy efficiency seeking to minimize the total power consumption of the data centre, and resource fragmentation seeking to merge small residual capacities spread on many physical machines into a chunk that is big enough for hosting incoming virtual machines. Based on data gathered by CactoScale and described in Infrastructure and Load Models, CactoOpt creates Optimisation Plans. These Optimisation Plans describe changes in mapping between virtual and physical machines as well as reconfigurations of physical infrastructure. The changes are later enforced by Virtual Middleware Integration (VMI). To cover all validation goals, we have prepared following scenarios, functional requirements and criteria.

a) SCENARIOS AND FUNCTIONAL REQUIREMENTS

Below four scenarios and functional requirements of CactoOpt are described. The scenarios list the steps that are taken by CactoOpt and must be verified to ensure a complete running of the CACTOS tools. Functional requirements detail the desired properties of Optimisation Plans.

SCENARIO 1 – RESOURCE PROVISIONING (GOALS 1 AND 3 FROM TABLE 1)

1. CactoOpt is initialised and is triggered periodically, see Figure 1
2. CactoOpt configuration and optimisation goals are updated from the configuration files periodically allowing the administrator to change the goal in live operation. The configuration files include settings for placement, continuous optimisation (migration) and placement.

3. CactoOpt connects to the Runtime Model Repository

4. CactoOpt loads infrastructure and load models as well as historical Optimisation Plans from the Runtime Model Repository. The Infrastructure is heterogeneous with respect to available CPU cores, memory and storage. The models contain the information about power consumption.

5. According to the optimisation goal (see step 2), CactoOpt runs an optimisation algorithm from the suit of algorithms available to find a new optimal configuration for the datacentre.

6. CactoOpt generates a new Optimisation Plan

7. CactoOpt saves the Optimisation Plan to the Runtime Model Repository

8. CactoOpt disconnects from the Runtime Model Repository

9. CactoOpt informs VMI that new Optimisation Plan is available in Runtime Model Repository

10. Step one repeated

---

**SCENARIO 2 – POWER AND COST OPTIMIZATION (GOALS 1 AND 3)**

1. CactoOpt is initialised and is triggered periodically

2. CactoOpt configuration and optimisation goals are set to energy efficiency or consolidation.

3. CactoOpt connects to the Runtime Model Repository

4. CactoOpt loads infrastructure and load models as well as historical Optimisation Plans from the Runtime Model Repository. The Infrastructure is heterogeneous with respect to available CPU cores, memory and storage. The models contain the information about power consumption.
5. According to the optimisation goal (see step 2), CactoOpt runs an optimisation algorithm for either energy efficiency or consolidation from the suite of available algorithms to find a new optimal configuration for the datacentre.
6. CactoOpt generates a new Optimisation Plan
7. CactoOpt saves the Optimisation Plan to the Runtime Model Repository
8. CactoOpt disconnects from the Runtime Model Repository
9. CactoOpt informs VMI that new Optimisation Plan is available in Runtime Model Repository
10. Step one repeated

**SCENARIO 3 – EVENT DRIVEN OPTIMISATION (GOALS 6 AND 2)**

1. CactoOpt is triggered as a reaction on specific event (e.g. request to place a new virtual machine), see Figure 2
2. CactoOpt connects to the Runtime Model Repository
3. CactoOpt loads infrastructure and load models as well as historical Optimisation Plans from the Runtime Model Repository. Infrastructure has to be heterogeneous with respect to available CPU cores, memory and storage; Models have to contain the information about power consumption
4. According to the optimisation goal, CactoOpt runs an optimisation algorithm for either energy efficiency or consolidation from the suite of available algorithms to find a new optimal configuration for the datacentre.
5. CactoOpt generates a new Optimisation Plan
6. CactoOpt saves the Optimisation Plan to the Runtime Model Repository
7. CactoOpt disconnects from the Runtime Model Repository
8. CactoOpt informs VMI that new Optimisation Plan is available in Runtime Model Repository
9. Step one repeated

![Figure 2 Validation Scenario 3 for CactoOpt](image-url)
SCENARIO 4 – PREVIOUS OPTIMISATION PLAN IN PROGRESS

1. CactoOpt is triggered periodically, see Figure 3
2. CactoOpt connects to the Runtime Model Repository
3. CactoOpt finds that there is an Optimisation Plans IN PROGRESS

![Diagram of CactoOpt process](image)

Figure 3 Validation Scenario 4 for CactoOpt

FUNCTIONAL REQUIREMENTS

CactoOpt generates a new Optimisation Plan that manages resources taking into account:

1. The heterogeneity of infrastructure (Validates Goals 1 and 3)
2. The power consumption (Validates Goal 2)

b) CRITERIA

SCENARIO 1 AND 2

1. CactoOpt should start processing periodically with time interval periods defined by data centre operator.
2. CactoOpt should have read rights to infrastructure and load models as well as read and write rights to Optimisation Plans.
3. CactoOpt should read the configuration files periodically to make sure that the optimisation goals have not changed.
4. Loaded models should be available and consistent to the meta-model definition.
5. A generated Optimisation Plan should be consistent to the meta-model definition.
6. The Optimisation Plan should be readable by VMI.
7. Connection with the Runtime Model Repository should be closed.

SCENARIO 3

1. CactoOpt should start processing when request to place a new virtual machine appears.
2. Other points as for Scenario 1 and 2
SCENARIO 4

CactoOpt should abort optimisation until the next time interval.

c) FUTURE COMMERCIALISATION AND FEATURES

Most of the code for CactoOpt has been open sourced. UMU provides active support to any users of the developed code. In addition, CactoOpt will be further developed by UMU adding new features and algorithms, e.g., to support emerging hardware such as Rackscale computers. For commercialization, some of the autoscaling developed algorithms are planned to be integrated in the commercial offerings of Elastisys AB, a spin-off from UMU. Some of the continous optimization algorithms, and Causa are now the basis for a new spin-off from UMU.

3. CACTOSCALE

This section covers with detail the description of CactoScale, steps, criteria and results of its validation as well as planned future actions.

a) DESCRIPTION

CactoScale enables monitoring of cloud datacentres and provides CactoOpt and CactoSim with information on the status of the cloud platform using measurements from the compute nodes. The collected information can be distinguished into three categories; measurements from the physical layer, measurements from the logical layer and collected data logs. In the case of the physical layer we have data that can be collected by performing measurements on each physical host of the cloud platform. CactoScale can also extract information from the logical layer of a cloud platform. As the logical layer of the cloud we define all the metrics, data and services with regard to the virtual machines and the virtualisation middleware of the platform. Data logs are another source of information of a cloud for CactoScale. Data logs are valuable for a cloud operator because they enable the maintenance and support of the platform.

b) STEPS

In this section we describe four scenarios and functional requirements of CactoScale. The scenarios list the steps that are taken by CactoScale and must be verified to ensure a complete running of the CACTOS tools.

SCENARIO 1 – CLOUD PLATFORM DATA COLLECTION

This scenario describes the steps taken for data monitoring and collection. CactoScale is performing measurements on the nodes and stores historical data in HBase which is a scalable and distributed database.

1. CactoScale agents are initialised by setting desired sampling rate for adapters.
2. CactoScale collector is receiving agent data and parsing them based on the selected adapters.
3. CactoScale is storing historical data on HBase over time.

SCENARIO 2 – INFRASTRUCTURE MODEL GENERATION

CactoScale is collecting low-level data and storing them in a distributed database. The data are made available to CactoOpt and CactoSim by translating these data into the higher-level CACTOS model language. CactoScale is utilising a component named CactoScale model generator to create
model instances from the collected data and shares them by utilising a Runtime Model Repository. The steps of this process are the following:

1. CactoScale model generator is triggered periodically or by new monitoring data
2. CactoScale connects to the Runtime Model Repository if not connected
3. CactoScale generates or updates the infrastructure and load models
4. CactoScale disconnects from the Runtime Model Repository

**SCENARIO 3 – OFFLINE LOG ANALYSIS**
This scenario describes the steps taken to perform an offline log analysis of a strace log.

1. HBase schema tables (more details on the structure and architecture of the database can be found in D4.1 and D4.2) are imported to the database.
2. The log files are imported into HBase
3. Analysed strace data are stored in the HBase
4. Data are exported in csv format

**SCENARIO 4 – DATA ANALYSIS**
This scenario describes the steps taken to perform data analysis on the collected metrics.

1. Historical data are collected in the HBase
2. Data analysis is triggered periodically or on demand
3. SPARK performs correlation analysis using CactoScale (more details on the analysis are included in D4.3)
4. Results are exported in logs

**c) Criteria**

**SCENARIO 1 – CLOUD PLATFORM DATA COLLECTION**
The criteria for this scenario are to verify CactoScale’s ability to gather information on the physical nodes and the virtual machines based on measurements of CPU, memory, storage and network. The collection should not induce significant overhead on the monitored nodes or cause any disruption to their services.

**SCENARIO 2 – INFRASTRUCTURE MODEL GENERATION**

1. CactoScale should start model generation when certain time elapses
2. CactoScale should have read and write rights to infrastructure and load models
3. Generated infrastructure and load models should be consistent to the meta-model definition
4. Connection with the Runtime Model Repository should be closed

**SCENARIO 3 – OFFLINE LOG ANALYSIS**
The criteria for this scenario are to verify the ability of the tool to produce useful analysis from the strace logs. The tool should produce I/O activity measurements extracted from the log traces and exported in a compatible format such as a csv file. More details can found in D4.2.
SCENARIO 4 – DATA ANALYSIS

The criteria for this scenario are to verify the ability of the tool to produce useful analysis from data stored in HBase. The tool should produce pair wise correlation analysis logs.

d) RESULTS

SCENARIO 1 – CLOUD PLATFORM DATA COLLECTION

Scenario 1 was validated successfully. CactoScale collects data from various resources described in D4.1 using agents installed in the cloud testbed. The data are processed by the collectors in the testbed and stored in a distributed database. The database retains both historical data of the cloud testbed and a current snapshot. More details on the performance of these components can be found on D5.5, D4.4 and D4.3.

SCENARIO 2 – INFRASTRUCTURE MODEL GENERATION

CactoScale is able to perform successfully the series of actions listed below which are necessary for sharing monitoring data with CactoOpt and CactoSim. Future actions would be to further enrich the generated model instances completely with information on power.

1. CactoScale is triggered periodically or listening for new monitoring data
2. CactoScale is connected successfully to the Runtime Model Repository
3. Infrastructure and load models are created / updated by CactoScale model generation
4. CactoScale is disconnected from the Runtime Model Repository

SCENARIO 3 – OFFLINE LOG ANALYSIS

Performing analysis of strace log files is successful and the results of such analysis are further documented in D4.2. The data can be exported in a csv format using CactoScale tools.

SCENARIO 4 – DATA ANALYSIS

Performing analysis of metrics collected by CactoScale is successful and the results of such analysis are further documented in D4.3 and D4.4.

e) FUTURE COMMERCIALISATION AND FEATURES

CACTOS provides access to CactoScale in the form of an open source project hosted in Github, which is a popular open source platform for hosting source code. Technical support for further development will be provided to the open source community that is being formed based in their common interest to CACTOS project results. The future support carried out after the project will allow the tool to be a sustainable source which can be further utilised by any future projects that require tools and algorithms for cloud monitoring and data analysis.

4. VIRTUAL MIDDLEWARE INTEGRATION

The Virtual Middleware integration (VMI) is used by the CACTOS Runtime Toolkit to preform optimisation actions within the target cloud environment. The VMI has access to the Application Program Interface (API) of the target cloud and uses this to perform optimization actions when required. Currently a VMI has been developed for Flexiant Cloud Orchestrator (FCO) and OpenStack. The OpenStack version has been tested and operated with OpenStack Kilo. Yet, it only relies on the Compute (nova) API v2 which will also be supported by future versions of OpenStack nova.
**FCO**

This section will cover in detail the steps, criteria and results and future actions of the FCO VMI.

**a) DESCRIPTION**

The FCO VMI is used to perform the Optimisation Actions that are created by CactoOpt within the Flexiant testbed platform. This VMI connects to the target cloud by using the FCO API’s to perform these actions. The VMI Controller executes tasks defined by CACTOS on the data centre using a soap based FCO API or by directly accessing the underlying physical hosts using Intelligent Platform Management Interface (IPMI) to control actions. The main component, which handles flows between CACTOS and the FCO platform, is the runtime management. The VMI controller issue requests to the FCO API for controlling physical resources and power state of compute nodes in the Flexiant data centre. For application instance deployment, CACTOS uses Cloudiator, a multi-cloud application orchestration software. The VMI and the VMI Controller communicate with the Colosseum service to handle actions on application level.

The VMI controller component implements actions listed in the following table. The VMI controller handles these actions sequentially or in parallel, depending on set optimisation plan generated. The full list of actions can be found within Table 2.

<table>
<thead>
<tr>
<th>ACTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deletion</td>
<td>Trigger shutdown and deletion of a virtual machine in FCO.</td>
</tr>
<tr>
<td>Manage Physical Node</td>
<td>Powers off or boots up a physical compute node via IPMI. Also with the FCO API allows the placement of nodes into maintenance mode and running mode.</td>
</tr>
<tr>
<td>Migrate VM</td>
<td>Migrates a VM from one to another node in an Flexi data centre.</td>
</tr>
<tr>
<td>Scale In</td>
<td>Scales in a scalable application component (e.g. Frontend of a Dataplay application instance)</td>
</tr>
<tr>
<td>Scale Out</td>
<td>Scales out a scalable application component (e.g. Frontend of a Dataplay application instance)</td>
</tr>
<tr>
<td>Scale Memory</td>
<td>Enlarges or shrinks the memory of a virtual machine</td>
</tr>
<tr>
<td>Start VM</td>
<td>Starts a VM in suspend or shutdown state</td>
</tr>
<tr>
<td>Stop VM</td>
<td>Stops a running VM (shutdown state)</td>
</tr>
</tbody>
</table>

**b) STEPS**

The following list details the steps viewed as a scenario that are taken by FCO VMI and must be verified to ensure a complete running of the CACTOS Runtime Toolkit for a migration action.

**SCENARIO 1 – PERFORMING AN OPTIMISATION PLAN WITH A MIGRATION REQUEST**

1. VMI module connects to the Runtime Model Repository  
2. VMI loads the Optimisation Plan created by CactoOpt  
3. VMI authenticates to cloud platform before action is started.
4. VMI reads action and calls migration method.
5. Migration API method called, migration from host node to target node begins.
6. The execution status of the Optimisation Plan is marked as completed
7. VMI retrieves the next Optimisation Plan

c) CRITERIA
1. Authentication credentials are valid and the granted permission level permits actions to be executed
2. VMI module has access rights to Runtime Model Repository Optimisation Plans
3. The Optimisation Plan is readable by VMI.
4. Migration of VM begins.
5. Execution of Optimisation Plan has finished and the execution status has been committed to Runtime Model Repository
6. A new Optimisation Plan is created by CactoOpt and stored in Runtime Model Repository

d) RESULTS
The results of the migration scenario detail all criteria have been reached and the migration of a virtual machine can be achieved. The VMI module is able to connect to the Runtime Model Repository and read the Optimisation Plan successfully. This first authenticates with the target cloud platform and then is able to read the generated Optimisation Plan, extracting the target node for migration. The migration is then started and the Optimisation Plan is marked as completed and the next Optimisation Plan is retrieved.

e) FUTURE COMMERCIALISATION AND FEATURES
Future actions for the FCO VMI will be in the updating of functionality to allow additional actions to be performed such as looking at uses of the FCO API with different hypervisors actions such as suspending of VMs. This will be used by future optimisation algorithms that require further actions not already established. Also the full range of actions will be made available open source to quickly allow plugins to the FCO platform with the full VMI code made available open source.

OPENSTACK
In the following, the next sub-section describes the mode of operations and the functionality provided by VMI OpenStack. The sub-sequent sub sections introduce validation requirements for the VMI OpenStack as well as steps, criteria and results of these scenarios.

f) DESCRIPTION
The OpenStack VMI and OpenStack VMI Controller enable CACTOS functionality for an OpenStack-based cloud installations (cf. Figure 5). VMI enhances the default OpenStack REST interface to users so that they can also trigger application deployment. In addition, it intercepts calls to the standard OpenStack interface dealing with spawning and terminating virtual machines. The VMI Controller in turn executes tasks defined by CACTOS (mainly CactoOpt) on the data centre to enact optimisation decisions. The main component, which handles flows between CACTOS and an OpenStack cloud platform, is the runtime management.
In detail, incoming REST requests to VMI are translated from OpenStack format to a generic CACTOS format and then passed to the runtime management. CactoOpt as well as the runtime management component use the VMI controller to execute CACTOS generic tasks in an OpenStack specific way. The VMI controller may issue requests to the OpenStack REST API, or to one of the two proxy services for controlling physical resources and power state of compute nodes in the OpenStack data centre. For application instance deployment, CACTOS uses Cloudiator, a multi-cloud application orchestration software. The VMI and the VMI Controller communicate with Cloudiator’s Colosseum service to handle actions on application level.

The VMI Interface

The VMI provides two services over a REST interface that are interface compatible with the OpenStack API. They are used to intercept calls for creating and deleting virtual machines.
i. **POST to /<tenant-id>/servers**: Requests with valid (OpenStack) JSON payload will create and boot new virtual machines in OpenStack provided that CactoOpt was able to place them. The headers and JSON structure are compatible with OpenStack’s Compute API v2. The VMI translates requests from OpenStack specific to CACTOS generic format and forwards them to the runtime management. Here they are run against CactoOpt and then relayed to OpenStack (cf. Figure 6). The following JSON snippet shows a request as the nova command line tool would send it to the OpenStack API. The blue area shows that “cactos-testing” was selected as an availability zone, which—from a technical point of view—is an arbitrary availability zone. Despite the name, it is not linked to CACTOS functionality. The red area denotes meta-data passed to the virtual machine. In CACTOS this meta-data is in to describe grey-box applications. If the respective information is not provided, CACTOS will treat the newly created Virtual Machine as a black box application.

```json
{ "server":
  { "name": "colosseum-cactos-junku-Molpro-lccsd-kiz2016-09-27 23:35:47",
   "imageRef": "2f0cad2e-9495-4d5d-a3cc-70165ea665b6",
   "flavorRef": "8c68690c-f1ee-4329-b24b-d932e0d7cc4c",
   "key_name": "colosseum-cactos",
   "availability_zone": "cactos-testing",
   "metadata": {"molpro-input": "lccsd-kiz", "molpro-size": "950", "applicationType": "molpro-lccsd"},
   "security_groups": [{"name": "jclouds-colosseum-cactos"}]
  }
}
```

This JSON is then pre-processed by the CACTOS runtime management based on placement information retrieved from CactoOpt. The following listing shows the JSON that the CACTOS runtime management will then forward to the actual OpenStack REST server. The only thing that was changed is the bold part in the blue box. Here, the availability zone was specified more concretely down to the level of the physical machine the Virtual Machine should be placed on.

```json
{ "server":
  { "name": "colosseum-cactos-junku-Molpro-lccsd-kiz2016-09-27 23:35:47",
   "imageRef": "2f0cad2e-9495-4d5d-a3cc-70165ea665b6",
   "flavorRef": "8c68690c-f1ee-4329-b24b-d932e0d7cc4c",
   "key_name": "colosseum-cactos",
   "availability_zone": "cactos-testing:computenode02",
   "metadata": {"molpro-input": "lccsd-kiz", "molpro-size": "950", "applicationType": "molpro-lccsd"},
   "security_groups": [{"name": "jclouds-colosseum-cactos"}]
  }
}
```

ii. **DELETE to /<tenant-id>/servers/<server-id>**: Requests with a valid OpenStack server-id will trigger a shutdown and removal command on this virtual machine. This request is compatible with OpenStack’s Compute API v2 as well. The VMI translates requests from OpenStack specific to a CACTOS generic format and forwards them to the runtime management. From they are relayed to OpenStack. Delete requests do not contain any body, so that no code listing is available.

---

1 http://developer.openstack.org/api-ref-compute-v2.1.html
Besides, the VMI provides convenience methods in order to bootstrap entire applications. While this interface is not strictly necessary from a functional point of view, it eases access to the platform and also allows CACTOS to derive semantics about the application structure.

iii. POST to /applications/molpro: Requests with valid CACTOS JSON payload will create a new Molpro application instance. The JSON structure describes the hardware and the base image but also the Molpro input parameters. Requests result in a new Molpro computation being issued. The VMI uses Cloudiator’s Colossuem to deploy the application instance. The following listing shows such a valid JSON payload for a Molpro deployment:

```json
{
  "molpro_size": "950", "molpro_input": "lccsd-SP01",
  "idCloud": 1, "idImage": 38, "idLocation": 47, "idHardware": 13
}
```

The information in the red box is meta-data about Molpro that maps to the red boxes in the above OpenStack example. The information in the green box is deployment information specifying hardware, image, and location to use. This information is forwarded to and parsed by Cloudiator’s Colossuem. It is worth noting that the IDs used here do not map to OpenStack IDs, but to Cloudiator IDs. This means that the frontend used for deploying Molpro is independent from OpenStack and can be used with other cloud platforms as well.

iv. POST to /applications/dataplay: Requests with valid CACTOS JSON payload will create a new Dataplay application instance. The JSON structure describes the hardware and the base image for each component the application contains. The VMI uses the CAMEL language and Cloudiator’s Colosseum to deploy the application instance. JSON bodies for Dataplay are similar to Molpro bodies, but contain one element for each of the Dataplay components (Frontend, LoadBalancer, Master, PgPool, PostgreSql, Cassandra, Redis) and do not contain component-specific meta-data. The following JSON snippet illustrates the payload of a deployment command where one instance of all component is deployed and all instances use the same hardware flavour and the same image (Ubuntu 15.04).

```json
[
  "name" : "LoadBalancer", "instances" : 1,
  "idCloud": 1, "idImage": 22, "idLocation": 49, "idHardware": 17,
  "name" : "MasterNode", "instances" : 1,
  "idCloud": 1, "idImage": 22, "idLocation": 49, "idHardware": 17,
  "name" : "FrontendNode", "instances" : 1,
  "idCloud": 1, "idImage": 22, "idLocation": 49, "idHardware": 17,
  "name" : "PostgreSql", "instances" : 1,
  "idCloud": 1, "idImage": 22, "idLocation": 49, "idHardware": 17,
  "name" : "Cassandra", "instances" : 1,
  "idCloud": 1, "idImage": 22, "idLocation": 49, "idHardware": 17,
  "name" : "PgPool", "instances" : 1,
  "idCloud": 1, "idImage": 22, "idLocation": 49, "idHardware": 17,
  "name" : "Redis", "instances" : 1,
  "idCloud": 1, "idImage": 22, "idLocation": 49, "idHardware": 17,
]
```

As of now, application teardown is not being supported. Instead, an OpenStack user has to remove the virtual machines one by one either using the Dashboard (cf. Figure 4) or the command line interface.
CACTOS Proxying

The compatibility of the POST and DELETE services for virtual machines allows redirecting OpenStack user requests actually issued to the OpenStack REST API to the CACTOS VMI. This does not create a huge configuration overhead, as a typical OpenStack setup foresees a web proxy in front of the REST APIs anyway (e.g. for supporting HTTPS connections).

In our OpenStack testbed, an Apache2 webserver is configured as a proxy for the actual API. It redirects POST and DELETE requests for selected tenants via the Apache “redirect” module. The following list shows an example with the red lines referring CACTOS-specific configuration elements. Please note that \texttt{THE\_REQUEST} is a variable set by the Apache server and refers to the full HTTP request line.

Listen 8774  
<VirtualHost _default_:8774>  
Define CACTOS\_RUNTIME 10.1.1.150  
RewriteEngine on  
# Redirect VM POST requests to CACTOS VMI  
RewriteCond %\{(THE\_REQUEST)\} POST  
RewriteRule ^/v2/<tenant-ids>/servers$ http://\${CACTOS\_RUNTIME}:9090/services/<tenant-id>/servers [L,R=307]  

# Redirect VM DELETE requests to CACTOS VMI  
RewriteCond %\{(THE\_REQUEST)\} DELETE  
RewriteRule ^/v2/<tenant-ids>/servers/(.*) http://\${CACTOS\_RUNTIME}:9090/services/<tenant-id>/servers/$1 [L,R=307]
Features of the VMI Controller

The VMI controller component implements actions listed in the following table. The VMI controller handles these actions sequentially or in parallel, depending on specification from the calling component. The table also lists the enactor responsible for processing the calls. While in most cases, this is OpenStack through its API, some operations, e.g. for switching on/off virtual machines, requires interaction with the servers done over the IPMI-based Power Control Proxy component. Also assigning more or fewer physical resources to virtual machines is done via direct interaction with the compute nodes on the testbed and implemented in the Resource Control Proxy.

Table 3 Optimisation actions supported by OpenStack VMI

<table>
<thead>
<tr>
<th>ACTION</th>
<th>DESCRIPTION</th>
<th>FORWARDING TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deletion</td>
<td>Trigger shutdown and deletion of a virtual machine in OpenStack. This action is issued by the runtime management after processing a DELETE request from VMI.</td>
<td>OpenStack API</td>
</tr>
<tr>
<td>Initial Placement</td>
<td>Create a new VM on a named node in an OpenStack data centre. This action is issued by the runtime management after processing a POST request from VMI.</td>
<td>OpenStack API</td>
</tr>
<tr>
<td>Manage Physical Node</td>
<td>Powers off or boots up a physical compute node via the Power Control Proxy and IPMI</td>
<td>Power Control Proxy</td>
</tr>
<tr>
<td>Migrate Vm</td>
<td>Migrates a VM from one to another node in an OpenStack data centre</td>
<td>OpenStack API</td>
</tr>
<tr>
<td>Resource Control</td>
<td>Adapts the allocation of physical resources to a virtual machine for a VM on a compute node</td>
<td>Resource Control Proxy</td>
</tr>
<tr>
<td>Scale Disk Vm</td>
<td>Enlarges or shrinks the virtual disk of a virtual machine</td>
<td>OpenStack API</td>
</tr>
<tr>
<td>Scale In</td>
<td>Scales in a scalable application component (e.g. Frontend of a Dataplay application instance)</td>
<td>Colosseum API</td>
</tr>
<tr>
<td>Scale Out</td>
<td>Scales out a scalable application component (e.g. Frontend of a Dataplay application instance)</td>
<td>Colosseum API</td>
</tr>
<tr>
<td>Scale Memory*</td>
<td>Enlarges or shrinks the max. physical memory of a virtual machine</td>
<td>OpenStack API</td>
</tr>
<tr>
<td>Start Vm</td>
<td>Starts a VM in suspend or shutdown state</td>
<td>OpenStack API</td>
</tr>
<tr>
<td>Stop Vm</td>
<td>Stops a running VM (shutdown state)</td>
<td>OpenStack API</td>
</tr>
<tr>
<td>Suspend Vm</td>
<td>Suspends a running VM (pause state)</td>
<td>OpenStack API</td>
</tr>
<tr>
<td>Take Snapshot</td>
<td>Takes a snapshot of a running virtual machine</td>
<td>OpenStack API</td>
</tr>
<tr>
<td>Resume Snapshot*</td>
<td>Creates a virtual machine from a snapshot</td>
<td>OpenStack API</td>
</tr>
</tbody>
</table>
The Resource Control Proxy and the Power Control Proxy are necessary, as the OpenStack API does not provide similar APIs to access the physical resources of an OpenStack data centre. This is due to the fact that being a cloud middleware, it is focusing on the virtual/logical abstraction layer.

Both proxies are accessible only by CACTOS components. They both implement a security mechanism for authentication, and only forward authorized requests from outside to the data centre-internal network. The Power Control Proxy is named IPMI Proxy in Deliverable D7.2.2.

Actions marked with (*) require a specific kind of application to be running inside the virtual machine. In particular, the application has to be run as a daemon (or service) and has to start automatically when the virtual machine starts. If this is not the case, the operation will be executed successfully and the VM will be working, but the application inside it will not.

### g) VMIOS1 Deploying and Deleting a New Virtual Machine

Deploying new virtual machine based on decisions by CactoOpt is a core feature and core requirement of any CACTOS VMI. As discussed before, the VMI OpenStack intercepts the BOOT and DESTROY commands issued to the OpenStack REST service. Prerequisite for this scenario are a fully set-up data centre with OpenStack and CACTOS tooling enabled.

#### STEPS

Using the default OpenStack command line interface and Dashboard, the following actions shall be executed. (i) deploy a virtual machine flavour and image through any of the two interfaces. (ii) Once the VM has been started and is visible through both interfaces, it shall be torn down again.

#### CRITERIA

The general criteria are straightforward: First, it should not be necessary to change the user interface tools. Also, the accepted parameters and response messages shall be identical to the OpenStack standard tooling. Second, through the deployment process CactoOpt’s placement service shall be triggered and its results be forwarded to OpenStack. A Virtual Machine shall become visible in OpenStack (Dashboard or CLI). In addition to that, the VM should also become visible in the CACTOS Model Repository and the Historical Database of CacotScale. In case of failure (e.g. CactoOpt does not find a free node to place the VM), the new VM shall appear as a VM in error state in the OpenStack user interfaces (Dashboard, CLI). For tearing down a VM, results shall occur in reverse order: First, the VM’s state in the Model Repository shall be set to SHUTTING_DOWN. Then, the shutdown command shall be forwarded to OpenStack. Then, the VM shall be removed from the Model Repository and the snapshot view of the Historical Database. It shall remain in the history tables of the Historical Database and further its deletion shall be marked there as well.

#### RESULTS

The goals are met to a large extent only minor differences to the criteria showed up. First, VMs that cannot be placed do not show up in OpenStack in ERROR state, but instead an error response is returned which is compliant to the OpenStack API and eases error detection on client side. Second, Virtual Machines do not move in SHUTTING_DOWN, but is removed directly from RUNNING state. This functionality is a prerequisite for the larger and more complex validation scenarios described in Section IV.2.
h) VMOS2 MIGRATING A VIRTUAL MACHINE

This scenario covers the case where CactoOpt’s cyclic optimiser decides to migrate a virtual machine to another host. The VMI shall then execute the migration on a technical level.

STEPS

Starting from a pre-populated and CACTOS-enabled data centre, CactoOpt is switched off while VMI OpenStack and all other CACTOS components remain enabled. From this setting, several optimisation plans are generated manually such that some of them can be run on a technical level (i.e. hypervisors are compatible, CPU architectures are compatible, sufficient disk space available on target node, OpenStack overbooking factors are respected) and others are not (in particular, too little disk space is available).

CRITERIA

This scenario is evaluated positively when VMI OpenStack finds and picks up a new optimisation plan, picks the oldest unprocessed plan, and triggers the migration action. Then, all doable migrations shall succeed and all non-doable ones shall fail.

Here, succeed means that the status of the optimisation plan in the Runtime Model Repository shall first move to IN_EXECUTION. The state of the VM in the Runtime Model Repository shall move to IN_OPTIMISATION. While this state is set, the state shown in OpenStack is MIGRATING. Once OpenStack changes the state back to RUNNING, the state of the VM in the Runtime Model Repository moves back to RUNNING as well while the state of the migration action in the optimisation plan moves to COMPLETED_SUCCESS. After the migration has been executed, the migrated VM shall reside on the target node.

In contrast fail means that the status of the plan should move to IN_EXECUTION first and later to COMPLETED_FAILED. In between, the VM’s state in the Runtime Model Repository shall initially move to IN_OPTIMISATION, but later back to RUNNING. The VM shall remain on its old hypervisor.

RESULTS

All goals are met. This functionality is a prerequisite for the larger and more complex validation scenarios described in Section IV.2. This feature of CACTOS was successfully demonstrated at the M24 review.

FUTURE ACTIONS

While the basic migration mechanism works fine, we had to find out that the migration support in OpenStack is not as sophisticated as would be required for an automated as CACTOS. In particular, OpenStack Kilo does not offer any functionality to abort ongoing migrations. This feature is needed when a migration takes too long and this blocks CACTOS from accepting new user requests. Also, several timing issues inside OpenStack have been detected when stressing the system. Most importantly, security tokens exchanged between source and target node during migration may time out leaving the Virtual Machine to be migrated in an everlasting IN_MIGRATION state. According to the OpenStack documentation, these shortcomings were removed in later releases; e.g. in the Mitaka release. Also, the latest version of the API supports the abortion of migrations.
Upgrading the testbed and validating that these issues have been resolved is a major milestone for the time after CACTOS and before moving the technology to the RECAP and MELODIC projects and also before moving it to the bwCloud production environment.

**i) VMIO3 Switching On/Off a Physical Machine**

This scenario is not OpenStack-specific, but depends on the hardware configuration of the data centre. Nevertheless, we present it here as it is part of CACTOS’ OpenStack testbed.

**STEPS**

Starting from a pre-populated data centre, CactoOpt is switched off while VMI OpenStack and all other CACTOS components remain enabled. From this setting, optimisation plans are generated manually that switch off physical machines and later switch them on again.

**CRITERIA**

This scenario is evaluated positively when VMI OpenStack finds and picks up a new optimisation plan, picks the oldest unprocessed plan, and triggers the action. When switching off a node, it shall be set to `SWITCHED_OFF` in the Model Repository and also in the Historical Database. OpenStack user interfaces shall not list the host any more. No more updates of monitoring data shall be reported. When switching on the machine, the state shall be set back to `RUNNING` in the Model Repository and the Historical Database. Also, new monitoring data shall be issued once the machine has booted and OpenStack is able to deploy new Virtual Machines on that node. Also, the energy consumption of the data centre shall drop.

**RESULTS**

All goals are met. This functionality was successfully presented at the M24 review.

**j) VMIO4 Resource-Control on Hypervisor Level**

This scenario describes an experimental feature that is not OpenStack-specific, but depends on the operating system running on the physical machines (CentOS 7 in case of the UULM testbed). Nevertheless, we present it here, as it can be run in CACTOS-enabled OpenStack installations.

By default, when a physical machine is overbooked on CPU, the computing capacity is equally distributed among virtual cores. Using the resource controller, enables to assigned fixed, guaranteed quota of e.g. compute cycles to individual virtual machines prioritising (or disadvantaging) them compared to others\(^2\).

**STEPS**

Starting from a pre-populated data centre, CactoOpt is switched off while VMI OpenStack and all other CACTOS components remain enabled. From this setting, several virtual machines are moved to one hypervisor overbooking CPU capacity. Then, in all of the VMs high CPU load is generated, e.g. by

\(^2\) This feature is also described in D3.4. It is called “vertical scaling” there. We use a different terminology here, as OpenStack also supports vertical scaling, but denotes an entirely different action with that term (cf. in the Resource Control Proxy.

Table 3).
running the stress tool. Then, optimisation plans are generated manually that restrict compute resources of some virtual machines and not of others. While this is done, the cpu_steal metric per VM is monitored.

**CRITERIA**

This scenario is evaluated positively when VMI OpenStack finds and picks up a new optimisation plan, picks the oldest unprocessed plan, and triggers the action. When reducing the CPU resources of a VM, its cpu_steal values should increase. When increasing the capacity, the cpu_steal shall decrease. An increase beyond the number of virtual cores of that VM shall not yield any visible results.

**RESULTS**

All goals are met. This functionality could, however, not be made sufficiently stable to include it in the main CACTOS Runtime Toolkit. Nevertheless, it is available on github\(^3\). The tool applies cgroups mechanisms in combination with the CentOS 7 systemd architecture to realise its functionality.

**FUTURE ACTIONS**

Next steps are to integrate the feature in the main branch of the CACTOS Runtime Toolkit and roll it out over the entire testbed.

\( k \) **VM. OS5 APPLICATION DEPLOYMENT**

This scenario validates that the VMI is able to start new applications of the two CACTOS applications, Molpro and Dataplay.

**STEPS**

Starting from an empty data centre, the VMI OpenStack application deployment interface is invoked to start (i) a new Molpro LCCSD job, (ii) a new Molpro DFT job, and (iii) to start a new Dataplay deployment.

**CRITERIA**

For each deployed Molpro job a new virtual machine shall be deployed (cf. Section III.4.g). For each Dataplay deployment a set of new virtual machines is started and the application component instances wired to each other (cf. D5.2.2).

**RESULTS**

All goals are met. This functionality was successfully presented at the M12 (Molpro) and M30 (Dataplay) reviews. It is a prerequisite for the larger and more complex validation scenarios described in Section IV.2.

\( l \) **FUTURE ACTIONS**

Feature-wise, the VMI and VMI controllers for OpenStack have proved that they are able to support the requirements imposed by the CACTOS use cases and the other components of the CACTOS Runtime Toolkit. Therefore, the next steps will focus on an increase in the technology readiness level. This will take place by a restructuring of the source code to be more modular and by the establishment of unit testing for each of the

\(^3\) https://github.com/cactos/resource-controller
modules. In addition to that, an evaluation of the VMI the latest OpenStack version (Newton or Mitaka) is planned for winter 2016, before the first parts of the project move into production environments.
IV. VALIDATION BY USE CASE SCENARIOS

In this section, each scenario partner details their use cases and results of the final validation. Each scenario will be broken into distinct areas to be validated. For the business analytics this is represented as Business Case (BC) the scientific scenario as Scientific Case (SC) and Cloud Application case (CC).

1. FLEXIANT BUSINESS ANALYTICS

This section will provide the validation results of the Flexiant Business Analytics scenario. This will focus on two use case areas. These are known as Business Case BC.1 Node Load Distribution and BC.2 Node Fault Tolerance.

a) OVERVIEW

The CACTOS validation by FLEXI requires two components. First, FLEXI has set up and is running a Flexiant Cloud Orchestrator-based data centre and that has been enriched with the CACTOS runtime toolkit. Second, FLEXI has defined several scenarios from the perspective of a data centre operator. In the following, we present the results from the validation. The remaining sections discuss to what extent the FLEXI use case is currently supported.

b) NODE LOAD DISTRIBUTION

Summary of changes since D7.4.1: This use scenario was evaluated and most elements where completed as detailed within D7.4.1. Over the course of the project life time the full Cactos runtime tool kit has been deployed. This has resulted in a full run trough of this scenario using the multiple optimisation algorithms within CactoOpt.

DESCRIPTION

Node load distribution (BC.1) will be used to ensure that VM load is spread effectively across the cloud platform. To do this an Optimisation Plan will be taken from the Runtime Model Repository, this details optimisations that need to take place, such as the migration of virtual machines between nodes and stop and start existing nodes when required.

STEPS

These steps are used to validate the use case of the Business Analytics scenario. With these steps verified we will then look at the Business Analytics use cases.

The following list details the steps that are taken by the CACTOS tools for the Business Analytics use case and must be verified to ensure a complete running of the CACTOS tools.

1.) Visit https://cp.sd1.flexiant.net to verify the platform is reachable
2.) Verify exposed metrics required are exposed.
3.) Verify date/time stamp of exposed information is correct
4.) Verify HBase agent is running within cloud platform
5.) Verify HBase database is receiving data from CactoScale
6.) Verify HBase database sends pulled data into Runtime Model Repository
7.) Verify Data centre model is created within Runtime Model Repository
8.) Verify physical load model
9.) Verify Physical Data centre model
10.) CactoOpt runs
11.) Verify Optimisation Plan if a plan is created
12.) Calls FCO VMI
13.) Verify FCO VMI reads generated plan
14.) Verify actions are performed on cloud platform as stated by plan.
15.) Step one repeated

CRITERIA

Each step stated previously will need to meet set criteria to be counted as validated. The criteria of the Node Load Distribution require that the CACTOS runtime toolkit is able to start all required components and that an Optimisation Plan can be created when a high node load is detected on the platform. Finally the FCO VMI is called to perform the action to migrate VM’s from the node with high load to the required node. These have been split into 15 different criteria that need to be verified before the Node Load Distribution can be considered verified.

1.) https://cp.sd1.flexiant.net is reachable with no errors
2.) View full list of exposed information and is required
3.) Date/time stamp shows current date/time
4.) HBase agent running within platform
5.) HBase database has data from platform
6.) Verify HBase database sends pulled data into Runtime Model Repository
7.) Verify Data centre model is created within Runtime Model Repository
8.) Verify physical load model
9.) Verify Physical Data centre model
10.) CactoOpt runs and generates Optimisation Plan
11.) Verify Optimisation Plan is created when a node has a high node load
12.) FCO VMI is called
13.) FCO VMI reads generated plan
14.) Verify actions are performed on cloud platform as stated by plan.
15.) Step one repeated

RESULTS

As stated within the Steps section, a number of validation goals must be met. The steps detailed previously state 15 steps that need to be validated before the optimisation action can be validated.

Currently all criteria have been reached as all exposed data is currently being pulled into the platform and this is used to create a Runtime Model of the infrastructure that is stored within the Runtime Model Repository. The FCO VMI is able to execute the created Optimisation Plans within the FLEXI Testbed.
c) **Node Fault Tolerance**

Node fault tolerance (BC.2) is used to ensure that the cloud platform runs in the event of a node failure and identify bottlenecks around the handling of cloud-based faults.

**Summary of changes since D7.4.1:** This use scenario was evaluated and most elements where completed as detailed within D7.4.1. Over the course of the project life time the full Cactos runtime tool kit has been deployed. This has resulted in a full run trough of this scenario using the multiple optimisation algorithms within CactoOpt.

**STEPS**

Within BC.2 the following steps are used to validate the scenarios of the use case description.

The following list details the steps that are taken by the Cactos tools for the business use case and must be verified to ensure a complete running of the Cactos tools.

1.) https://cp.sd1.flexiant.net is reachable with no errors
2.) View full list of exposed information and is required
3.) Date/time stamp shows current date/time
4.) HBase agent running within platform
5.) HBase database has data from platform
6.) Verify HBase database sends pulled data into Runtime Model Repository
7.) Verify Data centre model is created within Runtime Model Repository
8.) Verify physical load model
9.) Verify Physical Data centre model
10.) CactoOpt runs
11.) Verify Optimisation Plan is created when a node has failed
12.) Calls FCO VMI
13.) Verify FCO VMI reads generated plan
14.) Verify actions are performed on cloud platform as stated by plan.
15.) First a shutdown node is started
16.) All VM’s on the faulty node are migrated
17.) Previous node is either shutdown or placed into maintenance
18.) Step one repeated

**CRITERIA**

Each step stated previously will need to meet set criteria to be counted as validated. The criteria of the Node Fault Tolerance require that the CACTOS runtime toolkit is able to start all required components and that an Optimisation Plan can be created when a fault is detected on the platform. Finally the FCO VMI is called to perform the action to migrate VM’s from the node if required then shut down this node once the migrations have been completed. To start a node then the VMI needs to be able to start an existing node that is currently either in a “STOPPED” state or in “MAINTENANCE” state. These have been split into 18 different criteria that need to be verified before the Node Fault Tolerance can be considered verified.

1.) https://cp.sd1.flexiant.net is reachable with no errors
2.) View full list of exposed information and is required
3.) Date/time stamp shows current date/time
4.) HBase agent running within platform
5.) HBase database has data from platform
6.) Verify HBase database sends pulled data into Runtime Model Repository
7.) Verify Data centre model is created within Runtime Model Repository
8.) Verify physical load model
9.) Verify Physical Data centre model
10.) CactoOpt runs
11.) Verify Optimisation Plan is created when a node has failed
12.) Calls FCO VMI
13.) Verify FCO VMI reads generated plan
14.) Verify actions are performed on cloud platform as stated by plan.
15.) First a shutdown node is started
16.) All VM’s on the faulty node are migrated
17.) Faulty node is either shutdown or placed into maintenance
18.) Step one repeated

RESULTS

As stated within the steps section, a number of validation goals must be met. The steps detailed previously state 18 steps that need to be validated before the optimisation action can be validated.

Currently all criteria has been reached as all exposed data is currently being pulled into the platform and this is used to create a Runtime Model of the infrastructure that is stored within the Runtime Model Repository. The FCO VMI is able to execute the created Optimisation Plans within the Flexi Testbed.

d) OVERALL RESULTS AND FUTURE ACTIONS

The conclusion of the validation for the business scenarios are as follows. Both highlighted scenarios have been completed and validated successfully. The Cactos runtime toolkit is able to achieve all actions required by pulling the required data from the FCO testbed and use a range of optimisation algorithms which can be changed by the administrator and allows the requirements of the use case to be reached.

Flexi will then look to taking forward key concepts of the use case development and improve upon the captured metrics and how this can be implemented for FCO customers.

2. UULM SCIENTIFIC COMPUTING

The CACTOS validation for the scientific computation scenario of UULM, is based on two pillars. First, UULM has set up and is running an OpenStack-based data centre as described in (D7.2.2 Physical Testbed) (D7.2.1 Physical Testbed). This data centre has been enriched with the latest release of the CACTOS Runtime Toolkit, as described in (D5.2.2 - CACTOS Toolkit Version 2). Second, in (D7.1 Scenario Requirements on Context-Aware Topology Optimisation and Simulation) UULM has defined several use cases from the perspective of a scientific computing engineer.

In the following, we present the results from the validation. The remaining sections discuss to what extent the use cases of our scenario are supported by the end of the project, but also sketches what upcoming actions are necessary in order to support the use cases in a commercial environment. The
described validation use cases are taken from (D7.3.1 Validation, Goals and Metrics.) and (CACTOS Consortium, 2016) respectively. We briefly summarise them here. As this document builds on intermediate results presented in (D7.4.1 Validation and result Analysis), each of the sections starts with a summary of changes since that point in time. These capture both changes of outcome and also changes of goals of the scenario where applicable.

\[a\] Validation of Deployment

**Summary of changes since D7.4.1:** This setting was validated positive for (D7.4.1 Validation and result Analysis). As it is a prerequisite to using Molpro with CACTOS rather than a feature of CACTOS, no updates have been incorporated since (D7.4.1 Validation and result Analysis). In consequence, the text below is identical to the text from (D7.4.1 Validation and result Analysis); only the version of the CACTOS Runtime Toolkit was changed to version 2.

**OVERVIEW**

This first use case checks that all tools can be installed as described and then run stable for a longer period.

**STEPS**

CACTOS Runtime Toolkit Version 2 is deployed as it currently is available in the project’s subversion repository (D5.2.2 - CACTOS Toolkit Version 2). CactoScale Runtime Model Updater, the Monitoring Cluster and the CactoOpt Cyclic Optimiser are also enabled and deployed. Additionally, a Molpro GUI is available, which contacts CactoOpt directly to get a location for initial VM placement. This GUI is used to continuously start Molpro computations.

**CRITERIA**

All tools run continuously without crashing. Monitoring data is stored at CactoScale and the Runtime Model Updater updates the models at the Runtime Model Repository accordingly. CactoOpt reads them from the Runtime Model Repository and outputs an Optimization Plan Model. The plan is picked up by the OpenStack VMI and then is executed. All steps are repeated on a timely basis.

**RESULTS**

As already described by (D5.5 Performance evaluation of the CACTOS toolkit on a small cloud testbed) the deployment of the CACTOS Runtime toolkit was successful. The installation of Runtime Model Repository, CactoScale (including Chukwa adapters, Chukwa agent, Chukwa collectors, the entire HBase cluster software), as well as CactoOpt, could be realised successfully.

In the early stages, crashes occurred due to concurrency problems when accessing the Runtime Model Repository. Most of these problems have been fixed and a continuous monitor-optimise-update cycle has been created for UULM’s OpenStack testbed. Crashes occur about once to twice a week and can be fixed by restarting the crashed component. This has been automated through cron jobs. This level of stability is acceptable for the outcome of a research project.
Also, from time to time, OpenStack enters an invalid state, e.g. when migrations take too long and internal tokens get invalid (cf. Section III.4.h))\(^4\). This requires manual intervention from an administrator.

In order to perform housekeeping, processed optimisation plans have to be removed from the Runtime Model Repository in order to ensure a better overview on what is going on in the system.

**FUTURE WORK**

Next steps will be to port the OpenStack VMI to newer releases of OpenStack and to verify if OpenStack’s stability has improved. In addition, the concurrency handling between CactoScale’s model updater, the VMI, and CactoOpt has to be hardened. We expect that the planned modularisation and unit tests (cf. Section III.4.l) will already lead to the needed results.

\(^b\) **SC.1 RUNNING MOLPRO IN A VIRTUALISED ENVIRONMENT**

**Summary of changes since D7.4.1:** This scenario was evaluated positive for (D7.4.1 Validation and result Analysis). We have repeated the validation for the latest version of the CACTOS Runtime Toolkit, which led to the same results. In consequence, the text below is identical to the text from (D7.4.1 Validation and result Analysis).

**OVERVIEW**

This use case checks if the Molpro application is prepared for being executed in a virtualised environment such as clouds. This first basic functionality then allows a deployment of Molpro in the available testbed.

**STEPS**

UULM deployed a virtual machine based on CentOS 7 where the Molpro binaries are installed. Then, an execution environment runs the application and awaits its termination. Once the application has been terminated, the results are compared with those of an execution on a bare metal machine manually.

**CRITERIA**

The application results of Molpro running in a virtualised environment shall be identical to the results from a bare metal machine.

**RESULTS**

The criteria have been met and Molpro can serve as an application for the scientific computing scenario. It is noteworthy that not all hypervisors might support the execution of Molpro, since this use case was only validated using kvm in UULM’s physical testbed (cf. (D7.2.1 Physical Testbed)).

\(^c\) **SC.2 DEPLOYING MOLPRO THROUGH CACTOS**

**Summary of changes since D7.4.1:** This scenario was evaluated positive for (D7.4.1 Validation and result Analysis). Based on feedback we received from the Molpro users, we adapted the behaviour

\(^4\) This behaviour is related to the bug described here, but happens due to a different cause: https://bugs.launchpad.net/nova/+bug/1599057
of the Molpro GUI and the deployment logic provided for Molpro. In particular, users did not want that CACTOS selects the sizes of the Virtual Machines automatically. In addition, they also did not want to lose control over the amount of memory reserved for the Molpro process (defining the ratio of memory available for the operating system and memory used by Molpro). As a consequence, we adapted the GUI to reflect the user needs and hence, also updated the steps and criteria for this scenario.

OVERVIEW
After SC.1 Running Molpro in a Virtualised Environment has validated the usage of the Molpro application in virtualised environment in general, this use case checks that Molpro applications can indeed be deployed using the CACTOS Runtime Toolkit.

STEPS
UULM created a Molpro-aware GUI for deploying Molpro applications on the OpenStack testbed using a Molpro-enabled virtual machine image. In addition, the GUI allows to select the amount of memory assigned to Molpro, and the size of the virtual machine. In order to ease demonstrability, the GUI comes with six pre-selected configuration files, the user can choose from.

CRITERIA
This validation is defined as successful when the CACTOS tools including the Molpro GUI present the following behaviour: (i) it is capable of assembling the input to an input file and for the purpose (ii) derive a virtual machine configuration from it. (iii) It can start a new virtual machine with that configuration. (iv) It is able to deploy Molpro on that virtual machine. (v) It can inject the input file in the virtual machine and start the computation.

For successful computations (i.e. sufficient hardware resources provided), the computation completes and the virtual machine is shut down after having notified the user about the success. For unsuccessful computations, the user is notified about the failure and the virtual machine shuts down as well.

RESULTS
All Criteria (i) - (v) have been reached. Hence, CACTOS provides an automated deployment loop suited for scientific applications.

For the initial placement of virtual machines, CactoOpt proposes a host based on constraints, depending on the type of Molpro job (e.g. DFT or LCCSD method (D7.3.2 Validation, Goals and Metrics)).

FUTURE WORK
In the scope of bwCloud (cf. (D2.5.3 Exploitation Activities Report)), the mechanisms demonstrated by the Molpro GUI shall be enhanced with support for multiple applications. In particular, it shall be used to support non-IT students starting applications needed for their studies.

d) SC.3 MONITORING MOLPRO INSTANCES
Summary of changes since D7.4.1: This scenario was evaluated positive for (D7.4.1 Validation and result Analysis). Nevertheless, improvements have been implemented since then. This section presents the results of the validation with CACTOS Runtime Toolkit 2. Changes were made to the
criteria as the monitoring of kernel API calls using strace proved to have a tremendous impact on performance.

OVERVIEW

This use case ensures that all the necessary monitoring information from Molpro is available. This information is required by CactoOpt to perform meaningful placement and run-time optimisation for the managed data centre.

STEPS

The first step for monitoring a Molpro instance is to have a running Molpro application as a result of SC.2 Deploying Molpro through CACTOS. The next step is to collect monitoring information about the deployed application during the whole execution time.

CRITERIA

The criteria as of (D4.2 Preliminary Offline Trace Analysis), require that the CACTOS Runtime Toolkit is able to start the application and install the necessary monitoring sensors. CactoScale is able to sample CPU load, the memory usage and disk consumption at least every 10 seconds. The samples must then be stored in a persistent way for further analysis. Further, the monitoring environment has to be able to periodically monitor disk access (e.g. via blktrace). Finally, it reports the termination of the application to the Molpro operator, e.g. via email.

The monitoring of access to the operating system has been proven impractical in (D7.4.1 Validation and result Analysis) due to a performance impact of >60% and is omitted in this version of the document.

RESULTS

The CACTOS Runtime Toolkit Version 2 is able to install and start the application (cf. (D5.2.2 CACTOS Toolkit Version 2)). Due to the overall monitoring architecture used in the UULM testbed based on kvmTop⁵, it is basically not necessary to install monitors inside the respective virtual machine. Instead, CACTOS uses information provided on hypervisor level.

A notification of the operator via mail is not supported and outside the scope of the project. For (D7.4.1 Validation and result Analysis), the Molpro GUI recognised the termination of the application by a callback and copied the results to its database and released acquired resources.

For this iteration, the enhanced deployment support (as described in (D5.2.2 - CACTOS Toolkit Version 2) as well as in Section III.4.k)) enables the deployment of Molpro-specific probes that support a dedicated monitoring of Molpro. Now, the application status is monitored and once the application has terminated the termination status as well as the output data is stored in CactoScale for later analysis. In addition, CactoOpt has been enhanced to query for completed jobs. In case such a job is found, the virtual machine is removed from the cloud.

Data sampling with intervals of 10 seconds and storing this data is possible both for the physical nodes and for the virtual machine. Monitoring of disk consumption (amount of free or used disk

---

⁵ https://github.com/cactos/kvmtop
e) SC.4 Mine Molpro Traces

Summary of changes since D7.4.1: This use scenario was evaluated positive for (D7.4.1 Validation and result Analysis). While we collected a larger set of log files collected through the project’s life time, we consider this scenario as outdated, as by M36 CactoScale supports online analysis. Hence, the following text is considered to be a reference and unchanged from (D7.4.1 Validation and result Analysis).

OVERVIEW

This use case exploits CACTOS’ monitoring and data mining capabilities to identify behavioural phases of the application and helps to derive a Molpro application model. However, it only considers offline analysis.

STEPS

UULM runs Molpro with the same virtual machine configuration, but on different physical nodes with varying hardware platforms (CPU, memory speed, etc...) and have the data stored in HBase.

CRITERIA

Based on the traces it is able to identify CPU and I/O bound phases of the application from the logs. The results provided by CactoScale identify how many CPU-bound and I/O bound phases an application run has had. It also identifies the portion the respective phases had on the overall run time and is able to identify their start and end times with respect to the starting time of the application.

RESULTS

Overall, all criteria have been fulfilled, but slight weaknesses in determining the number of phases have been identified.

f) SC.5 Prediction of Execution Time and Execution Phases

Summary of changes since D7.4.1: This use scenario was not evaluated for (D7.4.1 Validation and result Analysis). Since then new calibration runs for Molpro were made taking into account the size of the virtual machine, the number of cores of the virtual machine, the type of storage used on the physical machine (SSD vs HDD), the portion of virtual machine memory assigned to Molpro (and hence, also the portion assigned to the operating system for caching), and the input size of the computation. Based on these set of logs, Molpro’s behaviour was analysed. As Molpro’s input file support a Turing-complete language, they are subject to the Halting problem. For that reason, we gave up on a complete analysis of the input file, but focussed on the prediction of single DFT and LCCSD computations. For this document, “Overview” and “Criteria” were updated accordingly. “Steps” were defined and the “Results” are presented.

OVERVIEW

CactoScale analyses the input to Molpro jobs (e.g. DFT or LCCSD method (D7.3.1 Validation, Goals and Metrics.) and the amount of memory used by Molpro in contrast to the amount of memory space) is fully supported from Chukwa. Moreover, disk usage (read/write access and bandwidth) is also supported.
available for the virtual machine) and from it builds an instance of the application model capable of predicting the runtime of the application dependent on the hardware configuration used. It is also capable to estimate the number of CPU and I/O bound phases the application has and their order.

**STEPS**

Molpro applications are started with different input files and different hardware configurations. Then, the number of predicted phases and the predicted overall execution time is compared i) in isolation and ii) with contention on the compute node. The actual execution time is then compared with the predicted one.

**CRITERIA**

When a Molpro application is started, the runtime management queries CactoScale to derive a behaviour model. Regarding the number of phases, it shall predict the number CPU-bound phases and of I/O-bound phases with a precision of +/- 1. The estimated time for the phase length should on average be within a 10% interval of the real execution time.

**RESULTS**

This scenario produced mixed results. Molpro’s behaviour is deterministic in the non-contented case, so predictions of runtime match very well and the deviations are barely measurable. For DFT jobs (without I/O) and LCCSD jobs on SSD nodes, we measure a deviation in run-time of around 2 minutes with an average run-time of about 9 hours. For LCCSD jobs running on an HDD node, the deviation is around 20 minutes and the average run time 11 hours. The cause of the deviation is access to the disk. In consequence, the runtime prediction for CPU phases matches (and hence DFT jobs) in the order of sub minutes. Similar holds for I/O phases on SSD nodes whereas the length of I/O phases varies in the order of up to 5 minutes. As the number of phases is determined by the type of computations and we restricted ourselves to DFT and LCCSD, the number of phases is constant (see below).

In the contented case, however, results are very different. While running multiple virtual machines with DFT work load does not influence the behaviour and values from the uncontended case still apply, this changes significantly when mixing LCCSD workload with either DFT or other LCCSD workload. Alone the action of deploying new virtual machines on a physical node running LCCSD applications bears the risk of harming the performance of the application. This, however, is dependent on whether the LCCSD application is in an I/O intensive phase and on what steps OpenStack has to execute in order to deploy the new virtual machine. If the image to be used for the new VM is already available on that host, the impact is barely measureable. Otherwise, the need to copy and store it can harm both the performance of the running application (extending its runtime by up to 3h) and the deployment process: deployment is very slow (up to 2h) and causes OpenStack to timeout. Running two LCCSD jobs on the same host without any additional measurements did not lead to any results. We decided to abort this case after four days for two computations that would not take more than 10h each.

Regarding the number of phases, it is exactly 1 for DFT jobs. For LCCSD, the assessment of phases depends on the desired granularity. While we started with five phases in the beginning, more detailed analysis unveiled more than 30 very fine-grained phases. While the use of very few phases yields a better prediction of phase length, the use of more phases yields better results with respect
to the purity of phases. E.g. in the coarse schema, CPU-bound phases will unveil I/O behaviour to some extent which bears the risk of causing unintended interference of co-located workload. On the other hand, using more phases, however, reduces the opportunity to co-location contradictory workload (cf. SC.10 Phase-Aware Scheduling).

FUTURE WORK

Future work will concentrate on exploiting the insights learned from modelling Molpro runtime to other HPC-oriented use cases in the scope of bwCloud (cf. (D2.5.3 Exploitation Activities Report)).

g) SC.6 ONLINE PHASE DETECTION OF RUNNING APPLICATIONS

Summary of changes since D7.4.1: This scenario was not evaluated for D7.4.1. During the evaluation of SC.5 Prediction of Execution Time and Execution Phases it turned out that the length of phases is either deterministic in the case of uncontented operation and barely predictable in the contended case. The latter mainly due to the fact that the influence of OpenStack cannot be neglected, but is not reflected by CACTOS’ application models. For that reason, an online phase detection is not needed for Molpro and the scenario is abandoned.

h) SC.7 FAILURE DETECTION AND SNAPSHOTs

Summary of changes since D7.4.1: This use scenario was not evaluated for D7.4.1, as failures were not targeted in the first iteration of the project. In the second iteration, it turned out that the failure rates in the testbeds were too low in order to derive meaningful results from them. Also, it became apparent that the Molpro application cannot be restarted from a mere disk snapshot as offered by OpenStack. Instead, snapshots that include memory state would be required (as offered by e.g. kvm).

Therefore, only little effort was made to support the heuristic handling of compute nodes failure in the entire CACTOS chain. A mechanism to take snapshots of individual virtual machines has been implemented, though. For the sake of overview, the following text as well as “Steps” and “Criteria” are taken from D7.4.1 and sketch the original goals of the scenario, even though it has been abandoned. The “Results” paragraph reflects the status at M36 of the project.

OVERVIEW

CactoScale captures information in data centers and exploits the monitoring data to compute an average down time and meantime between failures for both individual nodes and over all nodes of the data centre. CactoOpt uses this information and the estimated execution time (cf. SC.5 Prediction of Execution Time and Execution Phases) in order to derive a snapshot interval for saving the application state. During runtime, it ensures that the snapshots are taken on a regular basis. For computing this interval, it takes into account how much time is needed to gather the snapshot.

STEPS

The monitoring component in CACTOS is aware of failures and provides information about failure occurrence to CactoOpt. CactoOpt triggers the creation of snapshots through the VMI.
CRITERIA

CACTOS Runtime Toolkit is capable of taking snapshots on virtual machine levels that can later on be manually deployed using the interface of the respective cloud platform. Side-constraints are that the average time required to take all snapshots is lower than the average time spent on re-computations due to failures. These average values should be taken over all runs of Molpro.

RESULTS

The development of failure detection has not been in the scope of the project, because the failure rate in the testbeds was too low. Therefore, failure detection also has not been considered in the second validation cycle. The models, however, support a snapshot action and the VMI OpenStack supports snapshotting of Virtual Machines.

FUTURE WORK

Larger data centres will have more compute nodes and hence the absolute number of failures will be higher. UULM plans to investigate failure handling techniques within the bwCloud project that will base its automation mechanism on the CACTOS Runtime Toolkit (cf. D2.5.3).

i) SC.8 APPLICATION RESTART FROMSnapshot IN CASE OF FAILURES

Summary of changes since D7.4.1: This use scenario was not evaluated for D7.4.1, as failures were not targeted in the first iteration. The capability to take snapshots of running Molpro applications enabled us to evaluate manually restarting a Molpro computation from a snapshot. This yielded the result that such a feature is not supported by Molpro, even if CACTOS would provide the restart mechanism. This insight was approved be the Molpro developers.

Therefore, his scenario was not further investigated for D7.4.2. Instead, we focussed on Scenario SC.9 Concurrent Usage Of Physical Resources.

j) SC.9 CONCURRENT USAGE OF PHYSICAL RESOURCES

Summary of changes since D7.4.1: This use scenario was successfully validated in D7.4.1. Since then, Molpro-specific scheduling algorithms have been implemented that e.g. prioritise SSDs over HDDs. Also, Molpro LCCSD jobs are never placed on disk-less nodes as this will kill both the performance of a job and the network to the network storage. For D7.4.2 the criteria and the results have been updated to reflect these refinements.

OVERVIEW

When deploying a new Molpro application CactoOpt takes into account the actual deployment situation and decides on the deployment location such that the resource requirements of the new application and the already running applications (each encapsulated in virtual machines) do not conflict. In particular, they shall not exceed the existing physical resources (e.g. free disk space or memory), because compute-intensive applications such as Molpro do not profit too much from overbooking.
 STEPS

Incoming Molpro computations are scheduled to available server hardware with respect to other running Molpro computations by CactoOpt and deployed there by VMI OpenStack. If no such hardware is available, the request is answered negatively.

 CRITERIA

When deploying multiple Molpro applications, no two LCCSD (high disk I/O usage) shall be deployed on the same physical node and the amount of virtual cores/memory shall never exceed the amount of physical cores/memory. Also, LCCSD jobs shall never be placed on nodes without physical local (i.e. scratch) disk and finally, physical nodes with SSD shall be prioritised over nodes with HDD.

 RESULTS

All criteria to schedule new Molpro applications are met successfully. Resource shortages on physical machines are handled in terms of reconfiguration, e.g. by migrating virtual machines; provided that a Molpro-aware optimisation algorithm is used by CactoOpt (cf. (D6.5 Final results from optimization algorithms validation and experimentation)). If no resources are available, a negative response is returned to the user.

 FUTURE WORK

The functionality required by and implemented for this scenario is mostly solid. However, under rare conditions the bootstrapping of new virtual machines or the execution of a migration can block for a very long time.

For bootstrapping this is the case when the following conditions hold for a dedicated physical machine: (i) there is a running LCCSD job on that physical machine; (ii) this LCSSD job is in a write heavy phase (not any I/O, but write), (iii) CactoOpt has scheduled a new job on the same machine (this has to be a DFT job), (iv) the latest deployment using the Molpro VM image was more than one hour ago so that it is no longer cached on that machine, but needs to be copied there.

For migration, the same circumstances lead to a similar behaviour when migrating DFT jobs to nodes where an LCCSD job is in an I/O intensive phase.

The migration performance will also suffer if the VM to be migrated is in a compute-intensive phase with a lot of access to memory (not cache). This is a well-known issue in hypervisor technology.

A solution to this for would be enhance the CactoOpt implementation such that it does not place new virtual machines on nodes that currently experience write-heavy I/O. Alternatively, it could reduce the resources assigned to the running resource-hungry process for the time the migration goes on. The latter has been evaluated using the Resource Control Proxy (cf. Section VMI.OS4 Resource-Control on Hypervisor Level), but has not been integrated in the final release discussed in (D5.2.2 - CACTOS Toolkit Version 2). As discussed, the Resource Control Proxy is available on github\(^6\), though, and the necessary actions are invoked by the VMI controller in case they appear in an optimisation plan. As said before, an integration of these features in the Runtime Toolkit is planned for winter 2016/2017.

\(^6\) https://github.com/cactos/resource-controller
k) **SC.10 Phase-Aware Scheduling**

Summary of changes since D7.4.1: This use scenario was not validated in D7.4.1 due to blocking SC.3 (missing monitoring data) and SC.6 (no online phase detection available). Since then, Molpro-specific scheduling has been realised in CactoOpt.

**OVERVIEW**

Use scenario SC.9 Concurrent Usage Of Physical Resources schedules the initial placement, while this scenario attempts to combine it with online scheduling. When applications have conflicting resource requirements and their application models further show alternating CPU- and I/O-bound phases, they are scheduled such that conflicting phases do never occur at the same time. When conflicts still occur (as prediction may not be fully accurate, cf. SC.5 Prediction of Execution Time and Execution Phases) the scheduler delays the execution of one or multiple applications such that the conflict does not become visible.

**STEPS**

For phase-aware scheduling the online phase detection (cf. SC.6 Online Phase Detection of Running Applications) is used together with the application and hardware dependent prediction of execution time (cf. SC.5 Prediction of Execution Time and Execution Phases).

**CRITERIA**

When deploying two Molpro applications in LCCSD mode to an empty testbed, the scheduler shall select the same multi-core physical node and start both applications. Their respective I/O phases shall never overlap. If such an overlap is determined, the scheduler (CactoOpt) shall delay/pause one of the two applications. The time required to execute both applications shall be lower than their successive execution.

**RESULTS**

Basically, CACTOS provides all technical requirements for realising this use case, in particular the SC.5 Prediction of Execution Time and Execution Phases, but also the possibility to priorities virtual machines using the Resource Controller (cf. Section III.4.j)). However, the realisation of this use case turned out to be more difficult than expected, because of two technical obstacles described in the following:

First, as described at SC.5 Prediction of Execution Time and Execution Phases, even though the phase prediction and detection work well in isolation, it is very hard to predict the effect of side effects that are caused by activity outside the control of CACTOS including the activity on disk triggered by OpenStack when a new virtual machine is started (what is the case when a second virtual machine is started). Such activity will cause a hard to predict extension of I/O-bound phases and therefore arbitrarily extend their time.

Second, while Molpro activities can be separated in I/O-bound and CPU-bound phases, this does not mean that CPU-bound phases do not perform I/O at all. They still perform I/O that only appear neglectable on the scale at which LCCSD jobs are running (6-10 hours), but actually manifest in write bursts of several minutes. If such a burst occurs while the second VM is an I/O intensive phase or experiences such as burst as well, the progress of both virtual machines will be tremendously
delayed. Depending on how bursts and I/O intensive phases interfere, the execution of both applications may still be faster than a sequential execution or not.

The fact that executing I/O inside a virtual machine introduces several layers of buffering (application -> operating system -> virtual device -> host operating system -> physical device) makes it hard to predict when load issued in the virtual machine will reach the physical device and cause congestion there.

FUTURE WORK

Future work has to investigate the subtle challenges that came to light while dealing with this scenario. One approach to deal with it in a generic, i.e. non-Molpro specific way, is to enhance CactoOpt such that it considers the I/O delay experienced by different virtual machines running on the same host and then reduce the allowed I/O bandwidth and CPU time for all but one virtual machine in order to reduce congestion. At UULM, a Bachelor student, Timo Müller, will further investigate this problem from mid-October on when working on his Bachelor thesis. A further master student, Santhosh Ramapuram, will work on the same problem, but with respect to network traffic. The insights gained in both theses, will be included in the CACTOS Runtime Toolkit as it is used in the bwCloud project (cf. (D2.5.3 Exploitation Activities Report)).

1) SC.11 DETECT LACK OF RESOURCES

Summary of changes since D7.4.1: This use scenario was unsuccessfully validated in D7.4.1 due to unsatisfied requirements to SC.3 (missing monitoring data) and SC.6 (no online phase detection available). Since then, the introduction of Molpro-specific monitoring tools allows closely following up with Molpro’s behaviour and CactoOpt’s Molpro awareness supports the recognition of lacking resources.

OVERVIEW

The lack of hardware resources can be determined by CactoScale when monitoring the application and the consumption of hardware. In particular consuming almost all of available disk space and having the application (excluding the operating system) consume almost all of the available memory give strong evidence that the resources are not sufficient. CactoOpt reacts to the findings of CactoScale.

STEPS

A Molpro application is started with not enough resources (e.g. a small disk). CactoScale monitors the shortage, CactoOpt can trigger vertical scaling to increase the disk up to a threshold with knowledge about the currently executed phase.

CRITERIA

When intentionally using a badly suited virtual machine for a particular application configuration (by by-passing CACTOS’ estimation capabilities) CactoScale shall notify CactoOpt about an upcoming failure when 99.99% of all available disk has been used. CactoOpt shall react by extending the virtual disk used by the application in case the physical node has more resources available. When no such resources are available CactoOpt triggers the creation of a snapshot of the application, terminates the virtual machine, and restarts the application from that snapshot on another physical node.
The lack of compute time is not an issue, as by definition of SC.9 no overbooking shall take place for Molpro-only data centres (and hence HPC-oriented CACTOS users).

RESULT

Similar as with SC.7 Failure Detection and Snapshots the application logic of Molpro does not support vertical scaling so a live upgrade of the running instance is not possible. Nevertheless, the detailed monitoring of Molpro’s resource usage in combination with sensors that parse Molpro’s log files (both provided by CactoScale) allow CactoOpt to determine a lack of resources, kill the respective virtual machine and restart the same configuration on a larger virtual machine.

We found out that a threshold of 99.99% of the available resources is not the best choice, as a too large consumption may leave the operating system in the VM with too little resources and which may lead to a stalling application.

In contrast, picking a fixed threshold of more than 1 GB of mandatory free disk space and more than 512 MB of mandatory free RAM yielded better results. Also, it has to be said that detecting lack of disk is easier than detecting a lack of memory, as from outside the VM (the default approach in CACTOS), only the overall RAM consumption can be measured which may yield imprecise results, as for I/O intensive applications, the operating system in the virtual machine will use the remaining memory as an I/O buffer. Due to this subtlety it is necessary to install additional probes inside the virtual machine to gather the required information.

In addition to that, the approach implemented in CACTOS cannot only be used to detect too small VMs, but also wrong configuration parameters chosen by the user and causing the application to crash. In that case, however, we do not trigger an automatic restart, but shutdown the VM instead.

We chose a shutdown instead of a termination as this allows the user to evaluate the reason of the failure.

FUTURE WORK

Unless the developers of Molpro change the way Molpro is implemented, the features currently shipped with CACTOS are fully sufficient to support Molpro and its envisaged usage. As described this happens at the cost of restarting the computation from scratch. Luckily, Molpro tends to reach the peak in the use of resources rather soon. After this point in time the resource consumption stays at that peak level and does not change a lot from there on.

m) SC.12 SIMULATE OF SPARE RESOURCES

Summary of changes since D7.4.1: This scenario was not changed since D7.4.1.

OVERVIEW

Running mostly Molpro in a testbed produces spare resources, which might be used for smaller applications not related to highly resource demanding chemical computations.

STEPS

CactoSim reads a snapshot of the data centre, provided by CactoScale and replays the load of a pre-defined time interval. CactoSim triggers interactions with CactoOpt interaction.
CRITERIA
The average of spare resources should be simulated for the testbed in this use case.

RESULT
This scenario has been successfully evaluated in (D6.5 Final results from optimization algorithms validation and experimentation), in particular in Section IV.1. While these scenarios do not exactly demonstrate the spare resources, they evaluate CactoSim’s support for simulating scientific computing applications collocated with other applications (Black Box) and with it the overall use of resources.

n) SC.13 SIMULATE CHANGE OF RESOURCE ALLOCATION
Summary of changes since D7.4.1: This scenario was not available in D7.4.1 and was only introduced in D7.3.2.

OVERVIEW
Simulating the effect of complex rearrangements by applying optimisation actions without applying them in real world offers a reasonable protection against misconfiguration and also yields faster results compared to trying them out in reality.

STEPS
As in SC.12, CactoSim reads a snapshot of the data centre, provided by CactoScale and replays the load of a pre-defined time interval. CactoSim triggers interactions with CactoOpt interaction.

CRITERIA
CACTOS can read a current state of the testbed and successfully applies optimisation actions from CactoOpt in a simulation executed by CactoSim.

RESULT
This scenario has been successfully evaluated in (D6.5 Final results from optimization algorithms validation and experimentation), in particular in Section IV.1.

o) SC.14 CONTROL POWER STATE OF PHYSICAL RESOURCES
Summary of changes since D7.4.1: This scenario was not available in D7.4.1 and was only introduced in D7.3.2.

OVERVIEW
A straightforward way of saving energy is to shut off physical resources when they are unused. Yet, with interactive data centres it might happen that specific physical resources become entirely void empty without a rearrangement or consolidation. Contrarily, when more resources are required, physical resources must be provided in a timely manner.

STEPS
Starting from an empty data centre with all but two virtual machines switched on, new Molpro virtual machines are started and run to the very end until the data centre is empty again. The amount of jobs will not overload the data centre in case all physical nodes are switched on.
CRITERIA
With more jobs being started, physical machines are switched on one by one until all physical machines are operational. With more jobs ending, physical machines will be switched off one by one until the starting scenario has been reached.

RESULT
CactoOpt has an integrated, configurable feature to switch on/off physical machines. When this feature is activated, the physical machines not extending the buffer size configured by the system administrator are switched off. When “2” is set as a buffer size, the criteria are satisfied.

This feature was shown in the year 2 review at M24.

FUTURE WORK
The outcome shows satisfactory results. For a production environment the aggressiveness of the algorithm should be adaptive based on the current system load. CactoOpt comes with such algorithms for autoscaling. These have to be evaluated for steering the power on/off strategy.

p) OVERALL RESULTS AND FUTURE ACTIONS
The conclusion of the validation for the scientific scenario as follows: Out of the 14 validation scenarios that were originally defined, two (SC.6 and SC.8) had to be abandoned. In one case, as the implementation of Molpro does not support the snapshot and restart functionality offered by state-of-the-art cloud platforms. In the other case, applying online phase detection proved to be not useful, as it is not necessary for isolated, non-contended settings and would require the modelling of the entire OpenStack platform for mixed, contended settings.

Another one (SC.7) is not fully applicable. While snapshots are supported as optimisation actions, Molpro cannot be restarted from them. In addition to that, the failure rate experienced in UULM’s OpenStack testbed was too low to derive meaningful results.

Amongst the remaining eleven scenarios nine are fully supported (SC.1, SC.2, SC.3, SC.4, SC.9, SC.11, SC.12, SC.13, SC.14). The two remaining ones will benefit from more work after the end of the project on the path to exploitation:

SC.5 has shown that the prediction of execution time for I/O intensive applications under congestion does need to take into account the behaviour of cloud middleware. This makes the optimisation algorithms and their underlying optimisation models dependant on the implementation of cloud middleware such as OpenStack and vulnerable to changes in their code base. Alternatively, cloud middleware could be implemented congestion-aware and a fixed amount of resources on each host be reserved for administrative purposes. Only once the operation of virtual machines does not interfere with the behaviour of the cloud middleware, can phase-aware scheduling be exploited to its full extent (SC.10). The results of the OpenStack evaluation are summarized in Table 4.

Table 4 Overview of OpenStack Validation Results

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>VALIDATION RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC.1 RUNNING MOLPRO IN A VIRTUALISED ENVIRONMENT</td>
<td>Succeeded</td>
</tr>
</tbody>
</table>
SC.2 DEPLOYING MOLPRO THROUGH CACTOS
Succeeded

SC.3 MONITORING MOLPRO INSTANCES
Succeeded

SC.4 MINE MOLPRO TRACES
Succeeded, but outdated due to available online monitoring data

SC.5 PREDICTION OF EXECUTION TIME AND EXECUTION PHASES
Succeeded. May benefit from further work on phase detection

SC.6 ONLINE PHASE DETECTION OF RUNNING APPLICATIONS
abandoned, as online detection not necessary in case of isolated scenarios and not possible in mixed scenarios

SC.7 FAILURE DETECTION AND SNAPSHOTS
failure detection not useful for testbeds due to too low failure rate. Snapshot supported as an optimisation actions. Yet, not applicable as Molpro does not support restarts from snapshots (cf. SC.8).

SC.8 APPLICATION RESTART FROM SNAPSHOT IN CASE OF FAILURES
abandoned as Molpro cannot be restarted from a snapshot

SC.9 CONCURRENT USAGE OF PHYSICAL RESOURCES
Succeeded

SC.10 PHASE-AWARE SCHEDULING
Technical foundations laid, but more work needed.

SC.11 DETECT LACK OF RESOURCES
Succeeded

SC.12 SIMULATE OF SPARE RESOURCES
Succeeded, as detailed in (D6.5 Final results from optimization algorithms validation and experimentation)

SC.13 SIMULATE CHANGE OF RESOURCE ALLOCATION
Succeeded as detailed in (D6.5 Final results from optimization algorithms validation and experimentation), Section IV.1 Run 6

SC.14 CONTROL POWER STATE OF PHYSICAL RESOURCES
Succeeded

Summarising, the outcome of CACTOS is a big stepping stone for operating small-scale HPC applications in off-the-shelf cloud middleware. Nevertheless, more work is needed in order to bring the desired features to a production-ready state. First, more features are required from the OpenStack middleware, including support for aborting migrations and congestion-awareness.

Second, the CACTOS experience has shown that each HPC application needs to be well-understood in order to operate them reliably and performant in virtualised environments. Here, the CACTOS tools provide valuable help and shall be applied to and extended based on other HPC applications as well.

These topics also point in the direction of our future work as described in (D2.5.3 Exploitation Activities Report). In the bwCloud project, we will work on making CACTOS tooling production-ready by exploiting its capabilities for hardware allocation based on user and application types. In the upcoming research project RECAP, UULM will continue the work on monitoring, deployment, application profiling as well as scheduling/optimisation and simulation together with UMU and DCU.

In the upcoming research project MELODIC, UULM will work on integrating CACTOS results with PaaSage results and applying them to big data cloud platforms.
3. **DATAPLAY CLOUD APPLICATION**

The validation of the cloud application use case is centred around the goals to evaluate CACTOS’ capabilities of horizontal scaling as well as to evaluate its capability to support applications that connect across data centre boundaries.

### a) CHANGES SINCE D7.3.2

The validation of the cloud application use case is a joint effort of UMU (application modelling), UULM (testbed operation), and FZI (simulation, cf. (D6.5 Final results from optimization algorithms validation and experimentation, 2016)) based on the open source code of Dataplay. Due to this new set of stakeholders, the validation scenarios did change compared to (D7.3.2 Validation, Goals and Metrics). The main difference is that all of the new stakeholders consider Dataplay as a blueprint for a distributed, highly scalable cloud application and have a more general interest in how to operate, run, and simulate distributed applications.

The validation of the cloud application scenario compares to (D7.3.2 Validation, Goals and Metrics) as follows: (i) we kept use cases EA.1, EA.2, and EA.4 (now called CA.1, CA.2, CA.3); (ii) we dropped EA.3 about self-healing applications, that we consider out of scope for CACTOS; (iii) we introduced two new use cases CA.4 and CA.5.

In the following, we present the results from the validation. The remaining sections discuss to what extent the use cases of our scenarios are supported by CACTOS at the end of the project.

### b) CA.1 INITIAL SYSTEM SETUP

#### OVERVIEW

This scenario checks if Dataplay application is properly deployed using the CACTOS Runtime Toolkit.

#### STEPS

The cloud operator defines the application template that describes the application components and how they interrelate. Then, a user can use the API of the CACTOS Runtime Toolkit to select the virtual machine types (flavours) and image types attached to each of the components to trigger a deployment over the CACTOS Runtime Management API (cf. Section III.4.k)).

#### CRITERIA

(i) An appropriate number of virtual machines is created each with the appropriated hardware flavours attached.

(ii) Application components are installed on the virtual machines, instantiated, and wired.

(iii) CACTOS Runtime Models are created and reflect the state of the real application running in the testbed. In particular, they capture the current state of the application including the number of instances per component, in which virtual machines they are running, and the application performance.

#### RESULTS

All criteria (i) - (iii) have been reached. Hence, CACTOS supports the deployment of multi-tiered cloud applications. Figure 7) shows a screenshot of an early version of the CACTOS control centre GUI.
c) **CA.2 USER-DRIVEN OPTIMISATION**

**OVERVIEW**

Based on monitored data, a user shall be capable of identifying bottlenecks in his application.

**STEPS**

1. Starting from the configurations used in CELAR (cf. Table 5 Dataplay Configurations as Used in CELAR), we measure load on the system components when stressed with different request patterns and read/write ratio on the pattern.

   **Table 5 Dataplay Configurations as Used in CELAR**

<table>
<thead>
<tr>
<th>#masters</th>
<th>#frontends</th>
<th>#databases</th>
<th>#masters</th>
<th>#frontends</th>
<th>#databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

2. We analyse the monitored data using CactoScale and identify hotspots and performance bottlenecks.
3. Based in the observations, we change the configuration and adapt the application implementation.

RESULTS

Evaluation results unveiled that the database configuration using pgpool is the actual bottleneck in the system. This led to a change in the application architecture to not use pgpool and an improved configuration of postgreSQL to be able to cope with more load and less processing time. We also could see that the performance of the other components in the system (frontend, load balancer, redis, Cassandra) has only neglectable effects on the overall response time which is why no work was spent on them.

With the results of the evaluation, we are now capable of stressing the application up to the point where the network becomes the bottleneck (cf. Section IV.3.e) and the workload consists of 75% write requests and only 25% read queries. At that point only the number of master nodes has to be scaled. The remaining components can cope with that load with only one instance.

A technical result of this use case validation is the creation of a multi-threaded workload generator that is capable of stressing up to 100 Dataplay VMs.

d) CA.3 AUTO-SCALING

OVERVIEW

Optimisation of an existing system due to increase in demand where the user selects which autoscaler to use given the tradeoffs between the autoscalers discussed in D3.4 and the optimisation algorithms to use.

STEPS

i- A deployment of Dataplay is running with the components distributed across the cloud
ii- The administrator set the optimisation algorithms (Scaling, placement and migrations) in the CactoOpt configuration files.
iii- The workload generator starts with a workload profile capable of stressing the application to scale out the master component to up to 20 VMs.
iv- The workload reduces such that there are scale-ins.

CRITERIA

I- When the workload starts, CactosScale monitors the load and the load is pushed into the models.
II- CactoOpt is able to pick up the new load.
III- New VMs are spawned according to the picked autoscaling algorithm and the load levels.
IV- VMs are placed correctly according to the placement criteria
V- VMs migrate as load shifts or VMs fails
VI- When the load is decreased, the VMs are shutdown according to the correct usage

RESULTS

The validation scenario was run with multiple autoscalers. The results support all criteria. Yet, with the network being a physical bottleneck (cf. Section IV.3.e), we had to reduce the peak load to not trigger more than 10 master nodes.
**e) CA.4 DISTRIBUTING THE DATAPLAY APPLICATION ACROSS MULTIPLE SITES**

**OVERVIEW**

Many datacentre applications today run distributed across multiple DCs for, e.g., data privacy reasons where the DB should not be moved outside of a certain domain or country due to data privacy laws or security concerns. In these cases, it is interesting to what extent CACTOS can support applications that are only partially under its control.

**STEPS**

I- Deploy the different components in a distributed manner with the PostgreSql deployed in UMU, while the other components at the UULM cloud environment.

II- Configure autoscaling algorithms

III- Start workload on the application

**CRITERIA**

I- The Response Time of the application should remain within acceptable ranges even with the distribution across Europe. The average RT should never go above 2 seconds, and ideally should be below 2 seconds.

II- The number of dropped request is marginal.

III- The bottleneck of req/sec is the network bandwidth and not the

**RESULTS**

We were able to achieve both criteria as long as the load does not saturate all the VMs for long periods of time, e.g., if the load increases by very large steps that require adding a lot more master node VMs (more than 5 for example while there are only 5 master nodes available), the available master nodes saturate and breakdown if they stay overloaded for more than 5 minutes. This suggests that adding some rate limiting at, e.g., the Load Balancer for such applications can be very useful in order to preserve the application reliability. The boundary of how fast a new master node can be added is bounded by (i) the way the deployment scripts are written, e.g. is an image with pre-configured binaries installed or is the binary built on the fly, and (ii) by the time the cloud platform, e.g. OpenStack requires to transfer the image to the physical machine. While (i) can be influenced by the CACTOS operator to some extent, (ii) is also limited by the speed of the local network infrastructure.

Also, it became evident that the network bandwidth is a crucial factor in such a set-up. In our case, the maximum achievable number of master nodes is <10, enough for serving up to 100 req/sec. For scale out factors beyond this, the network becomes saturated, causing work to queue up at the master nodes until they break down. This holds true even if more masters are scaled out. The reason for this observation is due to the configuration of the UULM testbed. Technically, all compute nodes communicate with the outside world using an 1GB uplink that has a physical bandwidth limit of ~125 Mega Bytes per second. As it is a duplex connection, this sums up to ~250 BM/s. Figure 8 shows the network load observed in the data centre that was only occupied by Dataplay, hence the start level of 7 VMs in Figure 10, but only 1 master in Figure 9. With an increase of load (cf. Figure 9) the number of virtual machines and masters increases just as the energy consumption and the network...
utilisation that has it peaks above 300 MB/sec the majority of this traffic is to the database and from the database.

It is important to note, though that the 1GB uplink does not hinder the performance of the application, as also with a database running inside the datacentre the link capacity in the UULM testbed is limited to 1GB.
f) CA.5 APPLICATION LEVEL LOAD-BALANCING
ALGORITHM EFFECTS ON OVERLOADED APPLICATIONS

OVERVIEW
This use case evaluates the ability of CACTOS to achieve low response time and throughput regardless of the application level loadbalancing algorithm used. This aims to validate how generic are the optimisations done by CACTOS with respect to the application configuration.

STEPS
i- Deploy the different components of the application
ii- Configure autoscaling algorithms,
iii- Change the deployment scripts of the HaProxy Loadbalancer to leastconn loadbalancing
iv- Start an increasing workload on the application and monitor the scaling, RT, and throughput of the application.
v- Stop the workload, and reconfigure the loadbalancer to use round-robin
vi- Start an increasing workload on the application and monitor the scaling, RT, and throughput of the application.

CRITERIA
i. The Configuration of application level architecture does not affect the QoS achieved by CACTOS

RESULTS
Dataplay showed variable performance with the two configurations as the leastcon loadbalancing algorithm resulted in high overloads on some of the VMs. We note that, for other applications we have tested other than Dataplay, leastconn showed superior performance compared to roundrobin. This suggests that for future work, CactoOpt should include probes and/or directives to tune such important parameters based on the load.

g) OVERALL RESULTS AND FUTURE ACTIONS
The results of the Enterprise Application evaluation are summarized in Table 6. The conclusion of the validation for the cloud application scenario is that CACTOS supports many of the requirements of distributed multi-tier applications. Yet, we also see the potential for further improvements.

Table 6 Overview of Enterprise Application Validation Results

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>VALIDATION RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA.1 INITIAL SYSTEM SETUP</td>
<td>Succeeded</td>
</tr>
<tr>
<td>CA.2 USER-DRIVEN OPTIMISATON</td>
<td>Succeeded</td>
</tr>
<tr>
<td>CA.3 AUTO-SCALING</td>
<td>Succeeded</td>
</tr>
<tr>
<td>CA.4 DISTRIBUTION</td>
<td>Succeeded</td>
</tr>
<tr>
<td>CA.5 APPLICATION LEVEL</td>
<td>Failed</td>
</tr>
</tbody>
</table>
These include support for auto-scaling of distributed databases and improving the way we currently handle application level loadbalancing.
V. TUTORIAL: ADDING A NEW APPLICATION TO CACTOS

This section details how to CACTOS-ify a new application and benefit from the features CACTOS provides.

This section describes how to add a new application to CACTOS. As an example, we use the Opencast lecture recording software that is used in production at Ulm University. This section first introduces Opencast, its application structure, and a motivation use case. After that, Section V.2 describes how to prepare this application in five steps. Finally, Section V.4 lists where to find further information.

1. THE OPECAST APPLICATION

Opencast is a distributed application for Lecture Recordings. It is characterised by a very peaky workload and some very I/O affine application components.

Opencast is a distributed application for lecture recording. It is currently used by Ulm University to support both lecturers and students and to increase the quality of the e-Learning offerings. At Ulm university currently ten lecture halls are equipped with recording hardware and software. Per year, this leads to recordings of several dozen lectures and even entire courses. These are provided to students for preparation and post processing of lectures as well as for preparation of exams.

An expansion of the service is strongly desired, but currently the necessary human and technical resources are lacking. Many of the hardware devices bought for Opencast are idle during semester breaks and at night. Supporting more lecture halls would require more hardware to be available (thus also wasting more resources), as well as more human resources to maintain them.

a) APPLICATION OVERVIEW

Opencast\(^7\) is a distributed application that consists of four different Opencast components and in addition requires an ActiveMQ-based message queue, a MySql database, and a shared NFS storage backend. Database, message queue, and NFS are shared by the Opencast components to persist the application state (database), to exchange status updates (message queue), and to share video files (NFS) respectively.

The Opencast application itself consists of several different deployment assemblies. These are grouped into presentation node, worker nodes, admin nodes. Presentation nodes provide an interface for the end-users (mostly students) that they can use in order to watch and download recorded and post-processed videos. The admin node provides a management interface for the content providers platform operators. Here providers can schedule their lecture recordings and upload video recordings from other sources. Each of the videos is processed according to a selectable workflow that determines which operations are invoked in the video in what order. Platform operators get access to the full list of recordings including already processed ones, failed

---

\(^7\) [http://opencast.org/](http://opencast.org/)
ones, and those currently in progress. Further, the admin interface allows them to perform manual steps in the processing workflow such as reviews and quality control.

Table 7 Opencast component overview

<table>
<thead>
<tr>
<th>component assembly (category)</th>
<th>purpose</th>
<th>used in use case</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfs (infrastructure)</td>
<td>shared file storage including workspace and archive</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL (infrastructure)</td>
<td>shared database</td>
<td>Y</td>
</tr>
<tr>
<td>ApacheMQ (infrastructure)</td>
<td>even queue and message bus</td>
<td>Y</td>
</tr>
<tr>
<td>admin (Opencast)</td>
<td>interface for providers and operators</td>
<td>Y</td>
</tr>
<tr>
<td>presentation (Opencast)</td>
<td>interface for consumers</td>
<td>Y</td>
</tr>
<tr>
<td>worker (Opencast)</td>
<td>workhorse for encoding</td>
<td>Y</td>
</tr>
<tr>
<td>capture agent (Opencast)</td>
<td>records events</td>
<td>N</td>
</tr>
<tr>
<td>ingest (Opencast)</td>
<td>buffer for concurrently arriving recordings</td>
<td>N</td>
</tr>
</tbody>
</table>

The component assemblies are summarised in Table 7. Besides the ones described above, the table also lists the Opencast Capture Agent (CA): A CA runs in the fields and is responsible for recording videos. During recording, the CA buffers the video, and only after completion sends it to the admin node. In very large installations with many capture agents it is further possible to use Opencast’s ingest nodes that provide contact points for CAs and buffer the video on server-side before sharing it with the other Opencast components. This feature avoids network bottlenecks and overload situations for the admin node.

b) USE CASE DESCRIPTION: OPTIMISED_ENCODING

Opencast supports a huge variety of different features ranging from video recording in lecture halls, over video encoding and video rendering to manual online post-processing of recorded lectures, e.g. cutting the videos or merging the recordings of the lecturer with the recordings of the projector image.

With respect to desirable CACTOS support, we can state the following: (i) All operations running on a CA are outside the scope of a cloud solution, as these kind of operations are executed in the lecture halls and linked to physical devices there. (ii) The process of uploading recorded video streams to ingest nodes can saturate the network. However, the saturation in both CACTOS testbeds is caused by the bandwidth available for incoming network traffic at the head nodes. That means that adding more ingest nodes will not result in a higher upload capacity. Hence, while optimizing the number of ingest nodes may be beneficial in large and very large settings, it is not supported by the current testbeds. (iii) The process of encoding a video is way more resource-hungry than uploading a video. Due to that, this use case will focus on optimising the overall encoding throughput.
In consequence, the use case described in the following does not consider CAs and ingest nodes, but focuses on the encoding process. Besides those, we will use all components shown in Table 7. In order to create load on the system, we shall upload pre-recorded videos to the admin node.

2. CACTOS SUPPORT FOR OPENCAST

This section summarises the steps needed to enable CACTOS functionality for Opencast.

This section contains detailed instructions about how to make the Opencast application ready for being used with CACTOS. Section V.2.a) starts off with repeating the possible scenarios for integrating applications with CACTOS. From that, we conclude that Opencast is best integrated using a white-box approach, as this is the only way to integrate applications that need auto-scaling support. The sections following it, describe the necessary steps to prepare Opencast for white-box model execution. These include the creation of a white-box model template, the creation of deployment scripts, the sizing of virtual machines, and the creation of CACTOS-enabled sensors. Finally, Section V.2.f) introduces an evaluation scenario and summarises the actual evaluation results.

As a basic pre-requisite, the section assumes a configured and running CACTOS installation. The CACTOS website describes how to realise such a basic set-up.

a) CACTOS APPLICATION MODELS

An application can benefit from CACTOS in multiple ways. In brief, the support is available based on three kinds of application models that differ from each other dependent on how much is known about the application itself and its behaviour. The application models are black-box, grey-box, and white box applications.

---

8 http://cactos-cloud.eu/docs/quick-start-guide
BLACK-BOX APPLICATION

When an application is deployed as a Black Box model to a data centre managed by CACTOS, no in-depth knowledge is needed. VM tagging via a String identifying the application name may be used so that the platform can recognise instances of the same application. For CACTOS being able to make use of a tagged virtual machine, the users have to make sure that they add the right meta-data to the virtual machine. Here, the application name (“opencast”) and the component name (e.g. “ActiveMQ”) need to be put in the meta-data field. Figure 12 shows a screenshot of a Virtual Machine for Molpro that uses additional meta-data to identify the application configuration.

After that, CACTOS’ behaviour analysis component will use the combination of application name and component name to derive a behavioural model from past runs of the very same component and application and assume the component will be exposed to similar load patterns leading to similar behavioural patterns.

With this strategy, profiles such as CPU-intense, I/O intense, and network-intense can be determined automatically leading to the selection of the correct host and the co-location of non-conflicting VMs on the same physical host. It is important to note, however, that black-box models consider individual Virtual Machines only and lack information on the application structure of distributed applications consisting of multiple Virtual Machines.
USE AS GREY-BOX APPLICATION
Grey-box application models too, capture applications consisting of only one Virtual Machine. In contrast to black-box models, CACTOS does not derive application behaviour from other runs of similar Virtual Machines. Instead, the grey-box model provided by the CACTOS operator contains the necessary behavioural information. Due to the fact that the model is based on insights from the application code, it usually yields much higher precision and is more reliable than automatically generated black-box models. The Molpro application shown in Figure 12 is realised through a grey box model.

The meta-data structure used to represent application behaviour information in grey-box models is identical to the meta-data structure used in black-box models (cf. (D5.2.2 - CACTOS Toolkit Version 2, 2016)).

USE AS WHITE-BOX APPLICATION
Black- and grey box application models help CACTOS in optimising the placement of Virtual Machines over the data centre. Yet, due to their single Virtual Machine view, they cannot support the operation of a distributed application beyond this. This kind of support can be provided by a white-box model which includes application deployment (all Virtual Machines are started as a whole, i.e. all Virtual Machines at the same time) and auto-scaling of individual components (add/remove Virtual Machines as needed).

The use of a white box application model requires an understanding of what kind of components a distributed application consists of, what the purpose of the individual components is and how they communicate. When automatic horizontal scaling shall be used, it is further necessary to understand what components can be scaled horizontally and which ones cannot, e.g. due to architectural reasons, and which one should not be scaled, e.g. because the implementation of that component does not support scaling or it is known that this component is not on any performance-critical path.

Based on this information, an application-specific template has to be created that will be instantiated for each new instance of the application.
b) **Step 1:**

**Creation of an Opencast Whitebox Model Template**

In order to make CACTOS white-box capabilities accessible for a new application, a white-box model template needs to be created. Usually, this is the task of the cloud operator who wants to offer this application as a new service to his or her customers. The white-box model template of an application is initiated by CACTOS for each instance of the application that is deployed. It describes the components an application is built from, the communication structure of the application and the Virtual Machine images it needs to run. More technical details on white-box models are available in (D5.2.2 - CACTOS Toolkit Version 2, 2016).

![Whitebox Model Template](image-url)

**Figure 13 Screenshot from Eclipse of Opencast White-box Model Template**

Figure 13 presents a screenshot of the Opencast white-box model template as defined by the cloud operator. It shows the Virtual Machine images defined for all application components. The Composed VM Images describe what kind of services run on that Virtual Machine the interfaces they offer and the remote interfaces they require for communication. Further, the connectors are shown that link the application components to each other based on the offered and required services. The Scalable VM Image Connector plays a special role, as it denotes a scalable component. For an
Opencast installation this is the Worker component. The sizing of the individual virtual machines in subject to Section V.2.d).

c) **STEP 2:**

**CREATION OF ** **O** **PENCAST ** **D** **EPLOYMENT ** **S** **CRIPTS**

The white-box model template defines the overall structure of the application and also sets constraints towards the virtual hardware that can be used for each of the application components, the maximum size of the scaled out application, as well as the operating system images to be used. It does, however, not specify how the components are connected to each other and how the application components get installed on the virtual machine.

![Figure 14 Opencast Deployment Dependencies](image)

In order to capture this kind of information, CACTOS makes use of selected features of the CAMEL DSL\(^9\) that supports the specification of configuration and installation scripts. For a white-box application such as Opencast, the defined scripts (life-cyce actions in CAMEL terminology) have to ensure that from a blank operating system-only Virtual Machine Image, a running application component is started. Further requirements needed are the specification of a boot-order on the component instances.

---

\(^9\) [http://camel-dsl.org/]
The scripts for the Opencast life cycle actions have been released\(^{10}\) and support CentOS 7 as operating system. They are stored in the CACTOS configuration with one file per application\(^{11}\). The same files also capture constraints on the boot order (for the configuration files, refer to the CACTOS code repository\(^{12}\)). The application descriptors are read by the CACTOS runtime management and applied to the CACTOS configuration.

A graphical representation of the boot dependencies are shown in Figure 1. This shows that all opencast components rely on the infrastructure components database, message queue, and storage backend. If these are not available, the opencast components will fail and not recover from the failure. In consequence, all three infrastructure components need to be started (and be operational) before the opencast components. This can be done in parallel, as there are no cross-references between them. Similarly, the opencast components can be brought up in parallel as well.

**d) Step 3: Opencast Performance Modelling**

![Figure 15 User Screen for Creating a New Recording Event in an Opencast (Step 3: Selection of Workflow)](image)

In order to configure what shall happen with newly created videos, Opencast offers the capability of configurable workflows. A workflow is triggered when a new video is uploaded to either an ingest node or to an admin node. Following the workflow, the video passes through several processing stages that may include automatic steps such as encoding and images processing including features such as OCR and detection of slide transitions, but also manual steps such as cutting, review, and finally publishing. An opencast installation can have multiple workflows installed and the workflow to be used can be configured on a per-video basis.

\(^{10}\) https://omi-github.e-technik.uni-ulm.de/omi-cloud/opencast/tree/master/opencast-223/centos-scripts

\(^{11}\) This feature was not developed during the project, but after its end in order to make CACTOS easier to configure when used in production environments. During the project, all project-specific use cases were hard-coded and adding a new application would require to change the code.

\(^{12}\) https://svn.fzi.de/svn/cactos/code/integration/trunk/eu.cactosfp7.cloudiator/input/applications/
THE DEFAULT OPENCAST “FAST TESTING” WORKFLOW

Opencast ships with two pre-configured workflows. One of them, the “fast testing workflow” is a fully automatized workflow whose 13 steps do not require any human interaction. In addition, it contains a range of resource demanding operations for video encoding and text extraction. Table 8 lists the individual steps and a rounded average duration of each step. The resource-demanding steps are video encoding (steps 4 and 5 implemented through the ffmpeg tool\textsuperscript{13}), preview generation (steps 8 running ffmpeg and step 9 implemented in opencast), as well as text extraction and OCR (step 10 running tesseract\textsuperscript{14}). All of the steps are performed on a worker node. When multiple workers are available, the tasks are assigned to workers on a per-worker basis dependant on the current utilisation of the worker node.

Table 8 Overview of Default Opencast Workflow Steps with duration and maximum measured CPU demand for a typical 90 minute lecture recording and a 4-core worker with 8 GB of memory (recording incorporates only the recording of the slides as shown on the beamer screen).

<table>
<thead>
<tr>
<th>step title</th>
<th>step description</th>
<th>duration for a 90 minute video</th>
<th>CPU demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 defaults</td>
<td>Applying default configuration values</td>
<td>&lt; 1 s</td>
<td>10%</td>
</tr>
<tr>
<td>2 ingest-download</td>
<td>Ingesting external elements</td>
<td>&lt; 1 s</td>
<td>10%</td>
</tr>
<tr>
<td>3 inspect</td>
<td>Inspecting audio and video streams</td>
<td>~1 min</td>
<td>10%</td>
</tr>
<tr>
<td>4 compose</td>
<td>Encode video</td>
<td>~20 min</td>
<td>340%</td>
</tr>
<tr>
<td>5 compose</td>
<td>Encode video</td>
<td>~20 min</td>
<td>360%</td>
</tr>
<tr>
<td>6 image</td>
<td>Creating Engage search result thumbnails</td>
<td>~ 0.5 min</td>
<td>30%</td>
</tr>
<tr>
<td>7 image</td>
<td>Creating Engage player preview image</td>
<td>&lt; 0.5 min</td>
<td>30%</td>
</tr>
<tr>
<td>8 segment-video</td>
<td>Detecting slide transitions in presentation track</td>
<td>~ 10 min</td>
<td>260%</td>
</tr>
<tr>
<td>9 segment-previews</td>
<td>Creating presentation segments preview image</td>
<td>&lt; 1 min</td>
<td>100%</td>
</tr>
<tr>
<td>10 extract-text</td>
<td>Extracting text from presentation segments</td>
<td>~ 6 min</td>
<td>350%</td>
</tr>
<tr>
<td>11 publish-configure</td>
<td>Publish to preview publication channel</td>
<td>&lt; 2 min</td>
<td>30%</td>
</tr>
<tr>
<td>12 publish-engage</td>
<td>Publish to Engage</td>
<td>&lt; 4 min</td>
<td>20%</td>
</tr>
<tr>
<td>13 archive</td>
<td>Archiving</td>
<td>&lt; 1 sec</td>
<td>10%</td>
</tr>
<tr>
<td>14 cleanup</td>
<td>Cleaning up</td>
<td>&lt; 1 sec</td>
<td>10%</td>
</tr>
</tbody>
</table>

RESOURCE CONSUMPTION AND WORKER SIZING

In order to identify opportunities to scale and to understand the boundaries of scalability, it is necessary to run the application with different Virtual Machine configurations in order to identify bottlenecks in the application implementation. The selection of Virtual Machines flavours to evaluate the constrains can either be based on experience in operating the application, an earlier operation of the application using black-box or grey-box models, or on systematic experiments. For Opencast, we used the approach of systematic experiments and varied the number of cores per worker Virtual Machine and the number of video encodings going on in parallel. We had intended to

\textsuperscript{13} https://ffmpeg.org/

\textsuperscript{14} https://github.com/tesseract-ocr
also vary the amount of memory per worker, but noticed during the other experiments that memory consumption is not an issue (see below).

Table 9 summarises the required processing time for a single, isolated video depending on the number of cores. With the number of cores, we also varied the amount of available memory. Its consumption, however, does not increase and is unrelated to the number of cores. The second conclusion to draw is that using more than four cores in the worker does not lead to any additional benefit. While ffmpeg does spawn more when more cores are available, these do not achieve any additional increase in load and further not result in any additional performance gains.

Table 9: Execution time processing a single 90 minute video depending on the number of workers.

<table>
<thead>
<tr>
<th></th>
<th>1 core</th>
<th>2 cores</th>
<th>4 cores</th>
<th>8 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilisation</td>
<td>100%</td>
<td>200%</td>
<td>up to 360%</td>
<td>up to 360%</td>
</tr>
<tr>
<td>memory consumption</td>
<td>&lt; 1.5 GB</td>
<td>&lt; 1.5 GB</td>
<td>&lt; 1.5 GB</td>
<td>&lt; 1.5 GB</td>
</tr>
<tr>
<td>duration</td>
<td>~160 min</td>
<td>100 min</td>
<td>~60 min</td>
<td>~60 min</td>
</tr>
</tbody>
</table>

Table 10: Total Execution time of 1-4 parallel jobs on one worker with 4 cores

<table>
<thead>
<tr>
<th></th>
<th>1 job</th>
<th>2 jobs</th>
<th>3 jobs</th>
<th>4 jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>overall duration</td>
<td>65 min</td>
<td>107 min</td>
<td>200 min</td>
<td>356 min</td>
</tr>
</tbody>
</table>

Table 10 in turn shows the overall duration of up to four encoding jobs running in parallel. It shows that running two jobs in parallel is beneficial to some extent. While both jobs take longer than when run in isolation (both run ~107 minutes instead of 65 minutes), the overall time is less than with a sequential execution (~107 minutes instead of 130 minutes). This is due to the fact that all steps except 4, 5, 10 can run truly in parallel. Taking the numbers from Table 9, these phases sum up to 46 minutes per video with sequential processing and 19 minutes where both jobs run in parallel, yielding roughly the 107 minutes execution time from Table 10. Hence, running two jobs in parallel on one worker leads to a throughput gain of 40%. This gain decreases with more jobs running in parallel, possibly due to the fact that parallelisation is less likely with three jobs and synchronisation overhead increases. Overall, with three jobs, there is a slight throughput loss of around 2.5%. Running four jobs in parallel leads to a loss in throughput of about 37%.

CONCLUSIONS: DESIRED OPTIMISATION SCENARIO

When running OpenCast in practise, the goal is to have both throughput (i.e. process many jobs at the same time), but also low latency (process a single video as fast as possible to make it available to students as fast as possible). From a latency point of view, it is possible to handle two jobs per worker. From a latency point of view, however, each job should have its very own worker so that a maximum parallelisation is achieved. As another constraint, the worker nodes should have 4 cores.

From an economic point of view, there are no further constraints. Neither do any of the other opencast components require much CPU or memory capacity nor are there any specific demands towards networking.
e) **STEP 4: OPENCAST-SPECIFIC SENSOR**

In order to support the automatic scaling of applications, the application has to provide application-specific metrics in order to figure out the right point in time to scale. For Opencast, scaling out (adding a worker) should happen whenever there are more active (currently executed) workflows than workers. Similarly, scaling in should occur when there are less active workflows than workers. Obviously, the worker(s) that should be removed when scaling in should be the ones that are idle\(^{15}\).

While the number of currently active workers is accessible through the CACTOS models – because each Virtual Machine is started by CACTOS and hence known – the number of currently active workflows is an application specific metric that is not captured by CACTOS by default. Instead, a custom sensor needs to be implemented and deployed in addition to the application.

The sensor implemented for Opencast\(^{16}\) captures the amount of instantiated, non-terminated workflows through a REST interface offered by Opencast. In order to trigger its deployment together with the application, the application description file has to be enhanced with the specific sensor specification as shown in Figure 16. The sensor is installed with the admin component that steers the execution of the workflows.

```json
sensors:
  -
    scheduleTimeUnit: "SECONDS"
    scheduleInterval: 1
    className: "de.uniulm.omi.cloudiator.visor.sensors.matterhorn.MatterhornSensor"
    metricName: "MATTERHORN-ADMIN-REQ_TOTAL"
    isVmSensor: true
    isPush: false
    configuration:
```

Figure 16 Sensor configuration for Opencast’s Admin Component

f) **STEP 5: EVALUATION AND SIMULATION**

This section evaluates the results of the CACTOS-ification of Opencast. We first define an evaluation scenario based on the current setting found at Ulm University and then run both a simulation and a workload in practise.

**EVALUATION SCENARIO**

At Ulm University, currently 10 lecture halls are equipped with Capture Agents. In addition, 4 mobile Capture Agents are available that can be placed in one of the 12 other lecture halls or one of the 82

---

\(^{15}\) Even though Opencast can also deal with failed workers and would just re-issue such a processing step to a running worker after a while.

seminar halls. Lectures at the university are run in sync in six 2h blocks starting at 8.00 in the morning and ending at 20.00. Lectures take 90 minutes and are effectively placed in the 2h slot at a granularity of 15 minutes (hence, lectures in the 8-10 slot can start at 8.00, 8.15, or 8.30). When exactly a lecture is started depends on the lecturer, but traditionally, start time is quarter past.

Hence, in practise, not all Capture Agents are active in every single lecture block. Also not all lectures do end at the very same time. Technically, Capture Agents buffer the recording and send it to the ingest nodes (cf. Section V.1). This means that in the worst case, ten recordings arrive at an Opencast installation at the very same time and have to be processed before the next ten recordings arrive two hours later.

These constraints give us 90 minutes to process 10 recordings of 90 minutes length. In order to shorten the run-time, however, we use a scenario where we need to process 10 recordings of 50 minutes length within 50 minutes. Here, the numbers from Section V.2.d) adopt linearly. The processing of such a 50 minute video hence, takes around 32 minutes.

The blue dots in Figure 17 illustrate the arrival times for our evaluation scenario. A new request is added to the system every two minutes. The orange squares and the grey triangles illustrate the number of workers in the simulated and a real-world scenario respectively.

![Figure 17 Evaluation Workload, Simulated Scaling, and Real-world Scaling](image)

**SETTING UP A SIMULATION**

For the OpenCast scenario, we use simulation in order to validate that CACTOS is able to scale the number of workers based on the current number of jobs in the queue. In order to perform a simulation the following three steps are necessary: i) creation of a data centre model for the simulation; ii) creation of a load model that describes when which load is issued on the OpenCast application; iii) run the simulation and display results.

i. The creation of a data centre model captures the basic structure of a data centre, the physical servers, their wiring, the virtual machines in the data centre, and the applications they are running. Multiple approaches exist to create such a model, but we recommend to start with importing the data centre from the runtime model repository17.

ii. The behaviour model defines what kind of load is issued towards the OpenCast service, i.e. at what point in time new videos are being uploaded to the service. Together with the white-box model this information allows the simulation to compute the resource demand

of the application. This step is detailed in (D6.5 Final results from optimization algorithms validation and experimentation, 2016) on pages 24, 25 as well as page 48 and is not repeated here.

iii. Once the data centre model and the behaviour model has been defined, the simulation can be run (this is step is explained in detail in the CACTOS guide18). When the results are available, the load and utilisation curves can be illustrated (again, this is explained in-depth in the CACTOS guide19).

RUNNING THE SCENARIO IN THE REAL WORLD

Running the scenario in the real world is as straightforward as deploying the application (cf. Section III.4.k) and issuing new videos to the OpenCaST API once the deployment has been completed. This can for instance be done with a curl script as illustrated in Figure 18. The workload can be scheduled using tools such as cron20.

The orange squares in Figure 17 illustrate the scaling behaviour of the evaluation scenario when run in the CACTOS testbed. It shows that the real-world system behaves more inert compared to the simulation (grey rectangles). The real-world setting reacts slightly slower to spawning new Virtual Machines and also slower to completed encoding jobs. Due to that, it does also not scale back to one worker node after the peak load has been processed, but stays with three worker nodes when the second bunch of video files arrives.

EVALUATION SUMMARY

Overall, the evaluation shows that the simulation is able to predict application behaviour precisely and based on it to also simulate optimisation behaviour (scale outs in this case) as shown in Figure

---

18 https://cactos.github.io/docs/tutorials/cactusim-run-a-simulation/
19 https://cactos.github.io/docs/tutorials/cactusim-display-simulation-results/
20 https://linux.die.net/man/1/crontab
17. With the application running longer and more monitoring data being available, the precision of the simulation can even be increased over time.

Table 11 Comparison of different Opencast properties related to operation

<table>
<thead>
<tr>
<th>property</th>
<th>before CACTOS-ification</th>
<th>after CACTOS-ification</th>
</tr>
</thead>
<tbody>
<tr>
<td>monitoring</td>
<td>not done before</td>
<td>Constantly done, also on application level</td>
</tr>
<tr>
<td>frequency of adding new worker instances</td>
<td>done only when absolutely needed (once in six months) due to the high effort</td>
<td>possible in the order of minutes</td>
</tr>
<tr>
<td>effort needed for adding a new worker instance</td>
<td>two persons for two days for installation, one further person for hardware acquisition</td>
<td>automatically, no human interaction required</td>
</tr>
<tr>
<td>maximum recordings processible per day</td>
<td>capped by a fixed amount of 5 worker nodes</td>
<td>limited by the network capacity to the cloud infrastructure (5x10 GBit/s)</td>
</tr>
<tr>
<td>maximum latency per video</td>
<td>&gt;24h</td>
<td>no latency</td>
</tr>
<tr>
<td>energy consumption</td>
<td>dependant on number of workers (energy consumption widely independent of load)</td>
<td>dependant on number of encoding jobs (no energy consumption when no jobs are being run)</td>
</tr>
<tr>
<td>sharing of hardware resources</td>
<td>not possible</td>
<td>easily possible due to cloud environment and auto-scaling features</td>
</tr>
</tbody>
</table>

The evaluation also shows that running Opencast in a cloud environment is a promising way to go. Second, it shows that by using CACTOS, the current Opencast workload of Ulm University can be handled way more dynamically as well as with far less human interaction and fewer wasted resources than before (cf. Table 11). These savings in both effort and resources allow the university to shift budget from the operation of the Opencast service to the extension of the service. This will almost immediately allow an increase of recorded lectures and lead to more satisfied students and lecturers.

3. **Summary and General Recommendations**

Overall summary of this section and recommendations for the practitioners.

This tutorial presented how to create introduce CACTOS support for Opencast, a complex, distributed application for lecture recording. In particular, it showed how to define a white-box model for this kind of application starting from scratch and from there making the application accessible by CACTOS in both real-world environments and simulations.
The overall process comprises five steps ranging from defining the structure of the application, the installation scripts for the individual application components, the sizing of the different component Virtual Machines, the definition of the scaling scenario, and finally its operating in practise. Table 12 list the times needed for the individual steps which sum up to working 7.5 days. It is worth noting that the team doing the integration were familiar with CACTOS and unfamiliar with Opencast (be reminded that an experienced Opencast operators teams requires 4 person days to manually set-up a worker, cf. Table 11).

Using a team familiar with CACTOS and unfamiliar with the application for comparison, is reasonable, as CACTOS is intended for cloud operators who would offer more than one white-box application to their customers, leading to experience with CACTOS. However, usually the application operators are no experts with this application at the beginning of the task.

<table>
<thead>
<tr>
<th>step</th>
<th>name</th>
<th>duration [person-days]</th>
</tr>
</thead>
<tbody>
<tr>
<td>step 1</td>
<td>whitebox model</td>
<td>0.25 – 1.0</td>
</tr>
<tr>
<td>step 2</td>
<td>deployment scripts</td>
<td>1.0</td>
</tr>
<tr>
<td>step 3</td>
<td>performance modelling</td>
<td>0.25 – 2.0</td>
</tr>
<tr>
<td>step 4</td>
<td>application-specific sensors</td>
<td>0.5</td>
</tr>
<tr>
<td>step 5</td>
<td>evaluation/simulation</td>
<td>2.0 – 3.0</td>
</tr>
</tbody>
</table>

The unfamiliarity with Opencast led to a rather long duration of steps 1, 2, and 3. These can be drastically cut when knowledge about the application is available. If this is not the case, we recommend a different, iterative approach to CACTOS-ification that reduces lead time: By only fulfilling step 2, the application can be put in operation almost immediately, but without a white-box model. Instead, a black-box model can be used. Based on knowledge and monitoring data gained over time, this black-box model can be replaced by a grey-box model.

Enhancing the set-up with a custom sensor (step 4) provides more insight into application behaviour leading to a refined grey-box model. Additionally, such a procedure will generate the necessary data to complete step 3 en passant. Once this stage has been reached, enough experience is available to complete step 1 quickly, as behavioural data has been collected by and stored in CactoScale and can be accessed through the regular CactoScale tooling\(^\text{21}\). This reduces the overall working time to create white-box support for an application to around three working days, but increases the time period required to achieve it. Having said that, even once a white-box model has been created, monitoring data from CactoScale can be used for refining it.

\(^{21}\)https://cactos.github.io/docs/tutorials/build-and-run-spark-applications/
4. Screencast and Installation Guidelines

Summary of support material and videos.

During the life-time of the CACTOS project, a large amount of supplementing training and installation material has been created by the project consortium. This is available at the project’s github page\textsuperscript{22}. It contains amongst others, a quick start guide\textsuperscript{23} that leads to several in-depth explanations including an overview of the individual tools, typical usage scenarios, and finally a set of tutorials that cover CactoOpt, CactoScale, and CactoSim. The material is completed by several developer guides that simplify getting started with CACTOS for adopters.

The videos produced for CACTOS are available on youtube, and can be accessed through the CACTOS website as well\textsuperscript{24}.

\textsuperscript{22} https://cactos.github.io/
\textsuperscript{23} https://cactos.github.io/docs/quick-start-guide
\textsuperscript{24} https://cactos.github.io/demo-videos-final-review
VI. CONCLUSIONS AND FUTURE ACTIONS

Reviewing the validation of the use case scenarios proves that the CACTOS Runtime Toolkit had made good progress towards supporting the documented requirements for all the scenarios. Most of the validation use cases have been fully met and the CACTOS Runtime Toolkit development functionality has been improved. Detailed here both scenarios within the Business Use case have been achieved as well as the majority within the Scientific and Cloud application use case.

With the development completed, future work will focus on the commercialisation of the CACTOS tools after the project has completed. Partners will look to further development of features of individual tools as well as the use of these tools within future research projects & commercial entities.
REFERENCES