RELIABLE CAPACITY PROVISIONING AND ENHANCED REMEDIATION FOR DISTRIBUTED CLOUD APPLICATIONS

Accompanying Document for

Deliverable D7.1
Initial Simulation Platform

Work Package 7. Large Scale Simulation Framework

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme Under Grant Agreement Number 732667

Title:	Initial Simulation Platform
Author(s):	Sergej Svorobej (DCU), Divyaa Elango (DCU), Thang Le Duc (UMU), James Byrne (DCU), Theo Lynn (DCU)
Editor(s):	Sergej Svorobej
Reviewed By:	Jörg Domaschka (UULM), Paolo Casari (IMDEA)
Document Nature:	Other
Date:	31 December 2017
Dissemination Level:	Public
Status:	WORKING
Copyright:	Creative Commons Attribution No Derivatives License (CC BY-ND 4.0)

Revision History

Version	Editor (s)	Date	Change
0.1	Divyaa Manimaran Elango, Sergej Svorobej	22.06.2017	Initial document outline.
0.2	Sergej Svorobej	26.10.2017	Redraft of some sections and stub data population.
0.3	Sergej Svorobej	28.11.2017	Added diagrams and summary of use cases, plus added requirements table. Minor outline modifications. Added references.
0.3.1	Sergej Svorobej	29.11.2017	Added definition of terms, executive summary.
0.3.2	Sergej Svorobej	30.11.2017	Introduction summary, requirements introduction
0.3.3	Sergej Svorobej	08.12.2017	Expanded functional requirements section
0.3.4	Sergej Svorobej	11.12.2017	Added section describing system models and finished Infrastructure model description
0.3.5	Sergej Svorobej	12.12.2017	Section on Event Generator and outline of Simulation Engine section. Also added application model description and workload model section

0.3.6	Divyaa Manimaran Elango	12.12.2017	Added use-case description
0.3.7	Divyaa Manimaran Elango	13.12.2017	Added simulation tools description
0.3.8	Thang Le Duc	13.12.2017	Integration Plans (3.3), Optimisation Engine (3.3.1)
0.3.9	Sergej Svorobej	14.12.2017	Finished Simulation Engine section and Data Collection section
0.4	Divyaa Manimaran Elango	14.12.2017	Added use-case introduction and simulation tools overview
0.4.1	Divyaa Manimaran Elango	15.12.2017	Added BT site topology description
0.4.2	Sergej Svorobej	15.12.2017	Added experiment description, results and simulation performance sections.
0.4.3	Jörg Domaschka	18.12.2017	First review and feedback
0.4.4	James Byrne	20.12.2017	Addressing feedback
0.4.4	Sergej Svorbej	23.12.2017	Finished addressing 1st round feedback
0.5.0	Paolo Casari	27.12.2017	Second review
0.6.0	Sergej Svorbej	30.12.2017	Finished addressing 2nd round feedback
1.0	Jörg Domaschka	31.12.2017	Compilation of submittable version

Executive Summary

This report captures the simulation platform design and capabilities which are derived according to the project use case definition and an overview of existing cloud/fog/edge computing simulation platforms.

The initial RECAP simulation platform is designed to reflect the requirements derived from the behaviour of the systems presented in each of the use cases with following properties:

- Geographical distribution
- Hardware resource capacity
- Virtual resource capacity
- Network topology
- Deployed applications (services) performance
- User requests generation

The simulation architectural design reflects the needs of cloud/edge/fog computing in general and the individual needs of the presented use cases. The simulation platform includes several key components: *ExperimentManager*, *SimulationManager*, *ModelMapper*, *EventCoordinator* and *EventGenerator*. The *ExperimentManager* polls optimization, simulation, and event generator for available options and launches simulation instance with the listed parameters. Its main purpose is to gather all models and system configuration settings for the simulation experiment. The *SimulationManager* controls the simulation cycle using the *ModelMapper* to link RECAP Infrastructure Models (RIM) and simulation engine model elements. The *EventCoordinator* is responsible for simulation results serialization to a database storage and administration of optimization and failure events. The *EventGenerator* creates a distribution of failures or new Virtual Entity (VE) arrivals for the duration of simulation. It can also be extended to inject other types of events into simulation routine.

The developed initial simulation solution has been evaluated with a series of scaled up models to investigate the current base performance and to establish system requirements which are presented in this document at a high level. The next steps for the simulation framework focus on increasing the speed of simulation runs as well as decreasing resource consumption through proposed feature development. Integration of existing alternative simulation tools with the optimisation framework and the data collection framework increases functionality range of RECAP simulation platform allowing support requirements of the use cases and the target domain in general.

Definition of Terms

Terms	Description
Fog Computing	"Fog computing is a horizontal, physical or virtual resource paradigm that resides between smart end-devices and traditional cloud or data centres. This paradigm supports vertically-isolated, latency-sensitive applications by providing ubiquitous, scalable, layered, federated, and distributed computing, storage, and network connectivity." (Iorga, et al., 2017)
Edge Computing	Edge computing is a reference to the service location at the edge of the network. Such close proximity to the user ensures low latency service responses (Iorga, et al., 2017)
Cloud Computing	"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (Mell & Grance, 2011)

Abbreviations

Abbreviation	Full Name
DC	Data Centre
PM	Physical Machine
VM	Virtual Machine
BRITE	Boston university Representative Internet Topology gEnerator
JVM	Java Virtual Machine
MI	Million Instructions
MIPS	Million Instructions Per Second
PE	Processing Element
CPU	Central Processing Unit
RAM	Random Access Memory
HDD	Hard Disk Drive
SSD	Solid State Drive
NAS	Network Attached Storage
DES	Discrete Event Simulation

Table of Contents

Table of Figures

List of Tables

1 Introduction

Large-scale computing systems are today built as distributed systems, where components and services are hosted on geographically spread infrastructure and accessed remotely through clients and devices for reasons of scale, heterogeneity, cost and energy efficiency (Östberg, et al., 2017). In some systems, in particular latency-sensitive or high availability systems, components need to be placed closer to end-users in order to increase reliability and reduce latency, a style of computing often referred to as edge or fog computing. However, while recent years have seen significant advances in system instrumentation as well as data centre energy efficiency and automation, computational resources and network capacity are often provisioned using best-effort models and coarse-grained quality of-service (QoS) mechanisms, even in state-of-the-art data centres. These limitations are seen as a major hindrance in the face of the coming evolution of the Internet of Things (IoT) and the networked society, which are projected to significantly increase the load on networks and data centres (Östberg, et al., 2017).

With this increasing trend towards edge and fog computing, the aim of the RECAP simulator is to simulate large-scale scenarios (typically consisting of nodes at a scale of thousands) in the cloud, fog and edge computing space, in order to provide decision and control support for application and resource administration. This will be accomplished through the simulation of applications and application subsystems, simulation of infrastructure resources and resource management systems, and experimentation and validation of simulation results. The RECAP simulator and associated models will provide solutions and support for understanding and predicting the impact of remediation and compute resource allocation policies induced on workloads and QoS metrics as well as trade-offs for energy efficiency and cost within cloud, edge and fog computing scenarios, without harming the service level agreements (SLAs) of the users.

The RECAP simulator will make reproducible and controllable experimentation possible, aiding in identifying targets for the deployment of software components and in optimizing deployment choices prior to the actual deployment in a real cloud environment. Such simulation will be extensively employed in RECAP, to both simulate distributed cloud application behaviours, and to emulate data centre and connectivity networks systems. The complexity and size of the systems addressed are in themselves prohibitive for full-scale deployment, and will be studied at several levels in simulation. The RECAP simulator will assist the RECAP optimizer in the evaluation of different deployment and infrastructure management alternatives in terms of cost, energy, resource allocation and utilization, before actuating on real application deployments. This will require accurate output through the monitoring of the status of data centre infrastructure in order to provide an effective decision support instrument. This is of particular importance for understanding and managing trade-offs in edge and fog computing scenarios.

The document is organised in the following manner; Chapter 2 describes the requirements of the industrial use cases the presented simulation platform is addressing. Chapter 3 describes the high level architectural decisions that were made in putting simulation components together according to the previous requirements elicitation. Chapter 4 provides results of preliminary simulation platform evaluation through scalability tests. Chapter 5 concludes the work that was done to date and looks at the upcoming challenges and possible solutions going forward in the project.

The contents presented in this deliverable examine the use cases from the perspective of the requirements of the RECAP simulator. Broader information describing use cases within the context of the whole project can be found in the D3.1 (RECAP Consortium, 2017), in (Östberg, et al., 2017) and on the RECAP website (https://recap-project.eu).

2 Initial requirements

The simulation framework requirements are based on the description of the use cases compiled by the RECAP project partners. These use cases describe the challenges that the industry faces today, from technical and business perspectives, when adopting technology solutions spanning across fog, edge, and cloud layers. The project focuses on five such scenarios, namely "Infrastructure and network management" (Use case A), "Big data analytics engine" (Use case B), "Fog and large scale IoT scenario for supporting smart cities" (Use case C), "Virtual content distribution networks" (Use case D1), and "Network function virtualisation" (Use case D2).

The content presented in this deliverable examines the use cases from the perspective of the requirements of the RECAP simulator. Broader information describing use cases within the context of the whole project can be found in the D3.1 (RECAP Consortium, 2017) and in (Östberg, et al., 2017).

The following presents the high-level role that the RECAP simulation framework is envisaged to take across each of the 5 use cases. A list of requirements derived from the use cases ensues. These requirements are presented with a focus on the RECAP simulator:

- *Infrastructure and network management (Use case A):* For the use case A, it is envisaged that simulation will provide a detailed evaluation of the cost-to-benefit ratio in order to enhance infrastructure management. Predictions on the infrastructure model for better placement of resources across the nodes can be provided. It is also envisaged that simulation will support effective optimization policies on energy-aware provisioning at resources and capacity level. It can be used to study the impact of latency in the network in order to avoid shortcomings in terms of resource placement.
- *Big data analytics engine (Use case B):* The solution offered by Linknovate.com (use case B) in a distributed cloud architecture brings a number of challenges which could benefit from the use of simulation within the context of the RECAP solution. This makes it possible to evaluate cost vs benefit trade-offs for infrastructure management with supporting predictions for the better placement of services. Optimisation policies related to service placement in order to reduce network latency and meet user demands can be experimented with. Simulation can also aid in the evaluation of the efficiency of the optimization policies on dynamic on-demand service provisioning with respect to optimal utilization of resources.
- *Fog and large scale IoT scenario for supporting smart cities (Use case C):* In the context of use case C, the RECAP simulator can help support the development of an effective optimization policy on resource placement to enhance the system performance to maximize resource utilization. It can aid in helping to predict bandwidth occupancy in a distributed large-scale environment to provide a cost-effective model. It can analyse the occurrence of data delivery latency and help to improve the optimization policies on relocation of services.
- *Virtual content distribution networks (Use case D1):* Simulation can help validate the optimum solution to the network traffic model to provide cost-effective infrastructure management. It can provide predictions on the infrastructure model for better placement of resources, and can support the development of optimised placement policies to achieve optimal availability of service to user.
- *Network function virtualisation (Use case D2):* In this case, simulation can provide prediction related to infrastructure planning to achieve a reduced service down time. It can

evaluate optimization policies in terms of resource provisioning to meet high availability and high performance with minimal service outages.

2.1 Initial functional requirements

Referring to *D3.1 Initial Requirements* (RECAP Consortium, 2017), there are six high level functional requirements outlined, as well as two requirements specifically relating to planning and optimisation metrics for the RECAP toolkit as a whole. These requirements have a direct implication on the simulation platform, and are as follows:

- Automated application deployment and orchestration (see Section 7.2.1 of D3.1)
- Auto-scaling (see Section 7.2.2 of D3.1)
- Application runtime enhancement (see Section 7.2.3 of D3.1)
- Relocation of workload (see Section 7.2.4 of D3.1)
- Application monitoring (see Section 7.2.5 of D3.1)
- Error detection and remediation (see Section 7.2.6 of D3.1)
- Optimisation planning (see Section 7.3.1 of D3.1)
- Optimisation metrics (see Section 7.3.2 of D3.1)

Table 1 presents the role and requirements for the RECAP simulator from this set of initial RECAP requirements. Table 2 presents a summary of the specific derived requirements for the RECAP Simulator. Referring to Table 1 and Table 2, the simulation framework is required to create a representation of the fog/edge/cloud systems. Shared system models reflect geographical distribution of sites hosting hardware nodes which provide compute resources such as CPU, memory, storage (R702, R711). These sites are virtualised (R703) and interconnected (R713). Hardware virtualisation allows deployment and management of cloud applications and of their individual components, therefore simulation is required to reproduce the corresponding application models (R704) and to provide integration with the optimisation framework (R701). This integration with the optimisation framework is based on the shared structure of system models (R702, R703, R704, R705, R706) and the ability to enact optimisation actions within simulation experiment (R709, R710, R714). The cost of resources on different tiers of infrastructure varies and can be an important indicator for optimisation decisions. Adding cost models for alternative usage of resources should be also an option within the simulation framework (R718).

Table 1. Role and related refined requirements for RECAP Simulator with respect to "D3.1 Initial Requirements"

D3.1 Funct. Req.	Role and requirements for RECAP Simulator	Req. No.
7.2.1 Automated Application Deployment and Orchestration	Prior to the placement of a component, simulation can be used to evaluate optimisation algorithms. It can also be used to compare if the optimisation decision taken for placement is the correct decision. Short term and long-term effects can be analysed with respect to performance, capacity, and what-if analysis (what if a failure occurs for example).	R701, R702, R703, R704, R705, R706, R707, R708, R709,
7.2.2 Auto-scaling	Prior to the vertical or horizontal application scaling, firstly, simulation can be used to evaluate trade-offs between these two actions to make better decisions, for example: "Is it better to add more resources to existing virtual entities or add more instances instead?". Secondly, simulation can also be used to compare the effects of optimisation decisions on specific application and overall system performance with respect to auto-scaling.	R701, R702, R703, R704, R705, R706, R707, R708, R709, R710
7.2.3 Application Runtime Enhancement	On the fly reconfiguration of an application implies a change in application behaviour which manifests in resource demand change. The role of simulation is to support the orchestration decision through the evaluation of different possible deployment options in order to ensure that QoS requirements are satisfied adequately.	R701, R702, R703, R704, R705, R706, R707, R708, R709, R710
7.2.4 Re-location of Workload	In a complex system setup where user workload dynamically changes due to user mobility service deployment, efficiency is hard to estimate. Simulation allows for the previewing of potential success or failure of an optimisation action whereby the application or its component is moved to distributed sites following the service user geographically. In addition, compute nodes at the edge typically have less capacity due to special restrictions and simulation experiments can aid in ensuring that available resource capacity is sufficient both at the time of deployment and longer term.	R701, R702, R703, R704, R705, R706, R707, R709, R710, R711, R712, R713, R714
7.2.5 Application Monitoring	The RECAP monitored data serves as a definition for simulation inputs and outputs. One of the simulation inputs is the application model consisting of multiple distributed components. Each of these components, depending on the usage, has a specific resource demand footprint. Through monitoring application over a period of time at runtime the correlation between an application and compute resources can be captured as in a model that can be used to simulate such behaviour. The information on geographical request origin becomes a part of user mobility model that is used to simulate application users within specific time frame. Similarly, the historical information on failures occurring in the system can help to model and fine-tune the effects of failure events in simulation. Other monitored data serves as a requirement for simulation in terms of important metrics that need to be derived for the use-cases, as well as data against which simulation results will be validated in the future.	R702, R703, R704, R706, R707, R712, R714, R715, R716
7.2.6 Error Detection and Remediation	It is required that simulation is used for evaluation of error detection and remediation policies for the autonomous systems. The observed erroneous behaviour can be injected into the simulation experiment to firstly evaluate effectiveness of detection mechanisms, and secondly to evaluate remediation action effectiveness. The effectiveness can be measured primarily in the reaction speed, false positives and system SLA and QoS compliance.	R701, R702, R703, R704, R705, R706, R707, R715, R716
7.3.1 Optimisation Planning	It is required that the simulation output is maintained in the format of models that are also used for estimation and prediction of trends through the use of machine learning and other techniques. A common format means that simulation based experiment output timeseries data (such as resource capacity and utilisation) can be fed into the prediction component in the same way as it would be in the runtime environment. Such integration with a prediction component can allow for gauging how good the prediction patterns are as well as the consequences of false positives. Simulation can also execute experiments generated with certain synthetic conditions in mind that are based on extreme scenarios that were not necessarily available for prediction model training. For example, "what will happen in case of power failure or unforeseen user pattern? Can the system still operate within the specified QoS?"	R701, R702, R703, R704, R705, R706, R707, R709, R711, R712, R713, R714, R715, R716, R717
7.3.2 Optimisation Metrics	Simulation can provide a system behaviour estimation to find a balance between conflicting objectives of infrastructure provider and application. Saving costs of operations is an important objective for infrastructure provider, but from the service owner point of view, usually, the latency of the service is the most dominant factor. Optimisation decisions must find a balance in such situations that satisfy both parties. Simulation can assist optimisation in simulating few scenarios in parallel looking for the best optimisation decision under various auxiliary conditions.	R701, R702, R703, R704, R705, R706, R707, R709, R711, R712, R713, R714, R715, R716, R717, R718

In the simulator, the users of the system are represented as mobile entities. When the location of a user changes, this induces changes in the interaction both with the application and with the underlying network infrastructure, based on the proximity of each user to different network access points. Such dynamicity needs to be accounted for during simulation via a geographically distributed user model (R712) and corresponding network behaviour simulation (R713, R714). As in any complex system, failures are inevitable. This requires simulation to reflect real-world failure events by including failure models in the simulations (R715, R716).

Table 2: RECAP Simulator-specific derived requirements summary

Number	Simulation requirement
R701	Integration with optimisation component via API
R702	Shared data structure depicting physical infrastructure models
R703	Shared data structure depicting logical (virtual) infrastructure models
R704	Shared data structure depicting application models
R705	Shared data structure depicting application deployment and orchestration
R706	Shared data structure depicting application workload models
R707	Simulation of system behaviour over time under specified workload models
R708	Simulation of error/failure occurrence
R709	Application horizontal scaling support during simulation execution
R710	Vertical virtual resource scaling support during simulation execution
R711	Geographically distributed site model
R712	Geographically distributed user model
R713	Simulation of network paths
R714	Network path reconfiguration depending on geographic location
R715	Shared data structure depicting application failure model
R716	Simulation of application failures in the system
R717	Integration with the prediction component
R718	Cost model attached to different resource usage

The integration with the prediction component makes it possible to employ simulation as a tool to test forecast models that are used to trigger optimisations. Simulation integration is needed via an API (R717) by using simulated system capacity data via the shared system models (R702, R703, R704, R705, R706).

Taking these requirements into account, the following section gives an overview of the state of the art of cloud simulation tools.

3 Simulation of cloud computing state of the art

In recent years there are many new emerging technologies that can produce and process huge amounts of real time data over the Internet, leading to many challenges in the area of cloud computing such as a lack of system performance, security, reliability and capital cost affects (Shi, Cao, & Zhang, 2016). Thus in order to analyse and evaluate real-world scenarios, cloud computing simulation platforms have become ever more essential and indeed have increased in their usage.

Two published reviews of cloud simulation tools have been carried out this year lead by partners in the RECAP consortium and supported by partners in the CloudLightning consortium – "*A Preliminary Systematic Review of Computer Science Literature on Cloud Computing Research using Open Source Simulation Platforms*" (Lynn, et al., 2017), and "*A Review of Cloud Computing Simulation Platforms and Related Environments*" (Byrne, et al., 2017). A *Market Briefing* is also available (https://recap-project.eu/media/market-briefings/) which can be downloaded from the RECAP website.

The high-level motivation behind these reviews was to gain an understanding of the current state of the art in the area of cloud simulation, in order to aid the RECAP project (as well as the Cloudlightning H2020 project) from the following perspectives:

- Platform requirements / design decisions: At this stage in the RECAP project, a high level understanding of current state of the art is essential towards (a) making an informed decision as to which simulation engine and extensions should be utilised in RECAP, as well as (b) understanding the current feature set across all tools. Publishing these results across two articles in a leading conference in the area early in the project (CLOSER 2017) led to the opportunity to present the overview and gain feedback on RECAP design decision options from leading academics in the field.
- Exploitation / dissemination: Reviewing the cloud simulation field from an open source perspective enhanced the understanding of the cloud simulation market, which is important for both the RECAP and the CloudLightning partners involved in the reviews. For RECAP, a better informed choice can be made not just taking into account the feature sets of the tools, but rather taking into account current activity and support across simulators, with the correct choice potentially leading to the sustainability of project results and support from the community

This section provides a short overview of the published works[1], a high-level reasoning behind the choice of tool for the initial version of the RECAP simulator is provided with respect to the derived requirements from the use cases.

The chart (see Figure 1) represents a descriptive analysis of the literature as carried out by the partners of the RECAP consortium. The analysis was conducted on Cloud simulation research in IEEE Xplore Digital library across different publication outlets including Journals, conference papers, articles and magazines, see (Lynn, et al., 2017). Lynn et al., (2017) provide a detailed literature review across 256 papers from 2009 to 2016, which gives in-depth knowledge on the current research work using open source cloud simulation platforms. From the result of descriptive analysis, CloudSim appears to be the first published and leading discrete-event simulation platform. There is an increasing focus on the use of CloudSim for research in this domain: 78

[1] the interested party can read these articles in more detail through access to digital archives or by contacting the RECAP related authors.

identified publications are shown in 2015, up from 2 publications in 2009, mostly driven by CloudSim and related extensions.

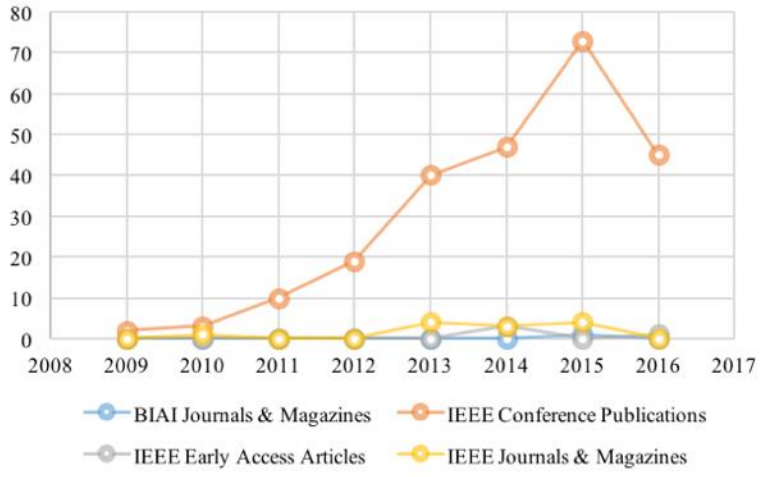


Figure 1: Cloud Simulation research using open source platforms by publication outlet and year (Lynn, et al., 2017).

Referring to (Byrne, et al., 2017) a high level and technical feature matrices for 33 discrete event simulation tools is presented providing a review on simulation tools, plugins and extensions. Table 4 lists identified tools that support DES for cloud computing, in alphabetical order.

Table 3: Identified Cloud Computing platform technical feature matrix, adapted from (Byrne, et al., 2017)

Simulation Platform	Language(s)	Platform Portability	Distributed Architecture	Model Persistence Type	Web API Availability	GUI Availability	Headless Execution	Result Output Format
Bazaar-Extension*	Java	Yes	No	Java classes	No	Yes	No data	Dashboard plots, F(X)yz 3D renders
CACTOSim	Java	Yes	No	Ecore	No	Yes	Yes	EDP2, CSV
CDOSim*	Java	Yes	No	Ecore	No	Yes	No	PNG export
CEPSim*	Scala, Java	Yes	No	Java classes	No	No	Yes	Text
Cloud2Sim*	Java	Yes	Yes	Java classes	No	No	Yes	Text
CloudAnalyst*	Java	Yes	No	XML	No	Yes	No	PDF
CloudEXP*	Java	Yes	No	No data	No	Yes	No data	No data
CloudNetSim++	C++	Yes	No	NED	No	Yes	Yes	Text
CloudReports*	Java, JS	Yes	No	SQLite DB	No	Yes	No	Javascript, text
CloudSched	Java	Yes	No	Text	No	Yes	No	XLS,Text
CloudSim	Java	Yes	No	Yaml	No	No	Yes	Text
CloudSimDisk*	Java	Yes	No	Java classes	No	No	Yes	XLS,Text
CloudSimSDN*	Java	Yes	No	CSV	No	No	Yes	CSV, JSON
CMCloudSimulator*	Java	Yes	No	XML, Java classes	No	No	Yes	Text
DartCSim*	Java, C++	No data	No	XML	No	Yes	No data	XML
DCSim1	Java	Yes	No	Java classes	No	No	Yes	Text
DCSim2	No data	No data	No	No data	No	No	No data	Text

DISSECT-CF	Java	Yes	No	Java classes	No	No	Yes	Text
EMUSim*	Java	No	No	XML	No	No	Yes	Text
GDCSim	C/C++, Shell	No	No	C code	No	No	Yes	Text
GreenCloud	C++, TCL, JS, CSS, Shell	No	No	TCL	Yes	Yes	Yes	Dashboard plots
GroudSim	Java	Yes	No	XML	No	No	Yes	Java API, Tracer handlers, Filters
iCanCloud	C/C++, Shell	Yes	No	NED	No	Yes	Yes	Text
iFogSim*	Java	Yes	No	JSON	No	Yes	No data	XLSX, PDF
MDCSim	No data	No data	No	No data	No	No	No	No data
MR-CloudSim*	No data	No data	No	No data	No	N/A	No	No data
NetworkCloudSim*	-	-	-	-	-	-	-	-
SimGrid	C/C++	Yes	No	XML, Java C++ classes	No	Yes	Yes	Text
SimIC	Java	Yes	No	text, Java classes	No	No	Yes	Text
SPECI	No data	No data	No	No data	No data	No data	No data	No data
TeachCloud*	Java	Yes	No	Java classes	No	Yes	No	Java graphs
Ucloud*	Java	Yes	No	No data	No	No	No data	No data
WorkflowSim*	Java	Yes	No	Java Classes	No	No	Yes	Text
CloudLightning Simulator	C/C++	No	Yes	JSON	Yes	Yes	No	JSON

**Derivatives or extensions of CloudSim*
[1] This refers to DCSim by Tighe, (2012)
[2] This refers to DCSim by Chen et al., (2012)

Table 4 summarises the technical feature for all 33 DES tools. Most of the tools are developed using the Java programming language. The majority of the tools are platform portable, meaning they can run in different operating system such as MS Windows or Linux. From the 33 DES tools, only Cloud2Sim supports a distributed architecture, and scales up for load balancing. In terms of model storage, Java classes are used by majority of the tools, followed by XML, NED, SQLiteDB, Text, C Code, C++ Classes, JSON, TCL and Yaml. Web APIs are used to control the simulation platform remotely, provided only by GreenCloud and CloudLightning Simulator. 16 tools provide a GUI. Most of the simulation tools support output in text format. The review also highlights the lack of support for parallel execution on a distributed system. *CloudLightning* (Cloud Lightning Consortium, 2017) is a parallel cloud simulation platform, which simulates hyper-scale heterogeneous clouds. It is highly scalable because it is written in C/C++, and has adopted MPI and OpenMP API for parallelization. The main objective is to provide improved service delivery, computational efficiency, improved power consumption and efficient and scalable use of resources. These identified technical features help structure and define the RECAP simulation platform technical features.

CloudReports (Sá, Calheiros, & Gomes, 2014) is a GUI tool that uses CloudSim as its simulation engine, and currently simulates the Infrastructure (Data centre, Host, User, Virtual machine) and generates a HTML report for each simulation.

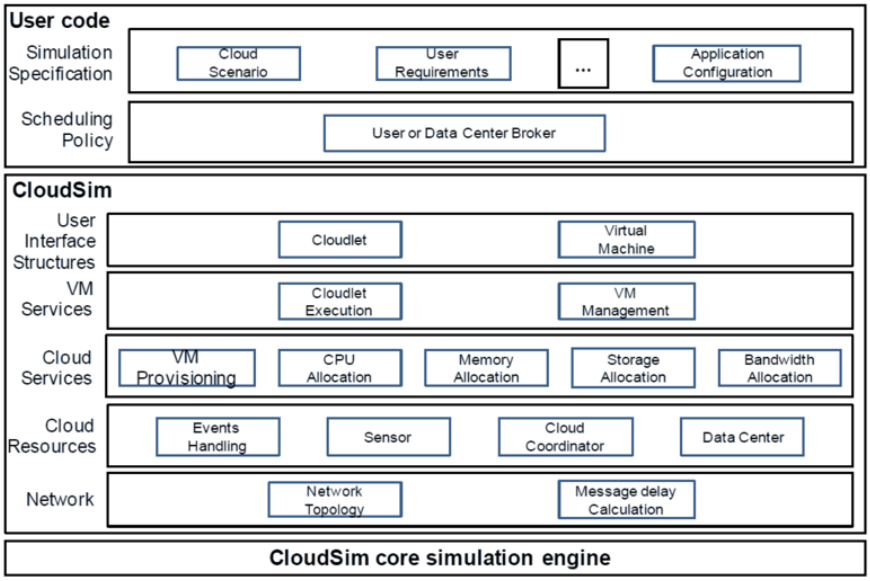


Figure 2:Layered CloudSim Architecture (Rodrigo N. Calheiros, 2009)

Based on this analysis, the decision made in RECAP is to utilise CloudSim as the base cloud simulation platform for delivery of the first version of the RECAP Simulator. The Cloudlightning Simulator is also under consideration due to its ability to run at speed and scale, and will be investigated for the next version of the RECAP simulator. It should be noted that Cloudlightning simulator has been released publicly as Open Source in December 2017, therefore detailed analysis as to the suitability of this tool to support the RECAP simulator has just begun at the time of completion of this document.

CloudSim extends and facilitates several requirements of the RECAP simulation framework which are described in Table 1. One of the main requirements is to create a cloud system representation. CloudSim supports seamless modelling and simulation of large scale cloud computing infrastructure which includes data centres, virtual machines, memory, storage, CPU, network-bandwidth, network-topology infrastructure, application and energy-aware models (Buyya, Ranjan, & Calheiros, 2009). Based on the requirements (R702, R703, R711, R713) the RECAP simulation framework should reflect the geographical distribution of sites, which can be supported by CloudSim using Boston university Representative Internet Topology gEnerator (BRITE) topology (Calheiros, Ranjan, Beloglazov, De Rose, & Buyya, 2011). The physical infrastructure model which contains geographical location and network configuration is stored in BRITE format (Medina, Lakhina, & Matta, 2001). CloudSim entities such as hosts, data centres, cloud brokers are mapped to a network node along with bandwidth and latency which helps to calculate network delay. The requirements (R704, R705, R706) represents the cloud application workload model and its deployment which is one of the novel features of CloudSim to utilize these models and implement Time-Shared and Space-Shared polices to see the effect of application performance. From Figure 2 Layered CloudSim Architecture the top most layer User Code has Application configuration which includes application type, number of task/job/cloudlet and its specification. Cloudlet class (Java Class in CloudSim) helps model the application by representing the computation complexity in terms of length, fileSize, outputSize and amount of data transfer. (Rodrigo N. Calheiros, 2009).

Requirements (R709, R710) represent resource allocation and application scaling during the execution of simulation experiment. CloudSim offers dynamic scaling of an application during the execution. The Cloud coordinator is an abstract class in CloudSim which helps in the application

scaling process. It periodically checks for the occurrence of load in multiple data centres and triggers the event for decision making during the simulation experiment (Buyya, Ranjan, & Calheiros, 2009). Requirement (R718) represents the inclusion of a cost model in the RECAP simulation framework. CloudSim has proven cost modelling techniques through its extended tool called CMCloudSimulator. CMCloudSimulator (Alves, Batista, & Filho, 2016) is a cost model simulator which helps to evaluate the best strategy for allocation and the best cloud provider to deploy applications in cloud for users to make effective investments.

The modelling of failures within the simulation models helps to realise the real-world system, which is one of the requirements (R715, R716). As CloudSim is an event-driven simulation experiment tool, it can simulate failures in the system. This is made possible by its extended feature set provided by FIM-SIM (Mihaela-Catalina Nita, 2014), a “Fault Injector Module”.

The next section presents the initial RECAP simulation framework design and implementation.

4 RECAP initial simulation framework design and implementation

The discrete event simulation (DES) based decision support process can be divided into three main phases: *modelling*, *simulation* and *results analysis*. During the modelling phase, the simulated system has to be defined by grouping interacting entities that serve a particular purpose together in a model. Once the representative system models are created, the simulation engine orchestrates a time-based event queue, where each event is admitted to the defined system model. An event represents actions happening in the system during operation time. Depending on the event type, the system reaction is being simulated, and metrics captured. These metrics are collected at the end of the simulation to form a base for results analysis, allowing for the study of system behaviour. Using DES is beneficial in a complex, large scale, non-deterministic system environment where the system definition via mathematical equations is no longer a feasible option. The fog/edge/cloud systems being focused on in RECAP are examples of such systems.

The aim of the RECAP simulator is to simulate large-scale scenarios in the cloud, fog and edge computing space, in order to provide decision and control support for application and resource administration. This will be accomplished through the simulation of applications and application subsystems, the simulation of infrastructure resources and resource management systems, and the experimentation with and validation of simulation results. The RECAP simulator and associated models will provide solutions and support for understanding and predicting the impact on physical resources, workloads and QoS metrics as well as trade-offs for energy efficiency and cost within cloud, edge and fog computing scenarios, without harming the service level agreements (SLAs) of the users.

Building on the requirements presented in Section 2, the following sections present the design and implementation of the RECAP simulator.

4.1 Design

Figure 1 presents a high-level design for the RECAP simulator, comprised of the following components: *Experiment Manager*, *Simulation Manager*, *Event Generator, Event Coordinator, Simulation Engine,* and *Model Mapper.*

The data flow in Figure 3 can be traced clockwise starting from the *Data Collection Framework* which provides an API to extract RECAP system models and system events based on monitoring data analysis. The *Experiment Manager* then passes all of the required information in the form of a *Simulation Experiment* to the *Simulation Manager* component which spawns a simulation process using an appropriate *Simulation Engine*. System models used in the RECAP project are different comparing to the models used by the core *Simulation Engine*, therefore a *Model Mapper* component is required to translate between simulation model entities and RECAP models. During the simulation process the *Optimisation Framework* is contacted periodically to check if any resource management actions needed. If the latter is true a new optimisation plan is sent back to simulation and enacted accordingly. After a simulation experiment is complete the results are saved for further analysis. Note that *Data Collection Framework* and *Optimisation Framework* are RECAP project services that are deployed within an operational fog/edge/cloud runtime environment.

The design for each of these and the role that they play in the overall simulation design is described in the following sections. As the current design is based on the initial requirements collected from the use-cases descriptions, these designs are expected to evolve with future project progression.

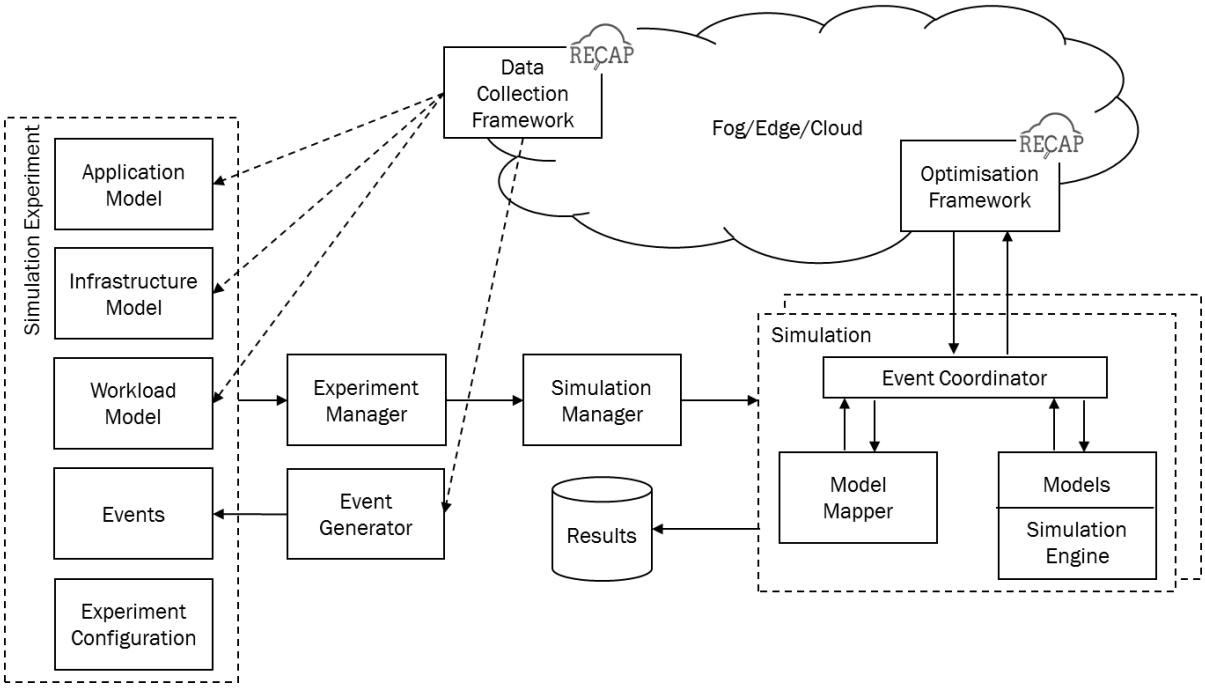


Figure 3: RECAP simulator high level design

Experiment Manager *(Related requirements: R701* R707*, R717)*

The *Experiment Manager* is a first point of contact for the user, and exposes a full list of functionalities of the *RECAP simulator* to the rest of the RECAP components. The main purpose of the *Experiment Manager* component is to serve as a hub that contains methods to gather the data needed to run simulation experiments. Through the provided API, the user can reach out to the different system components that an experiment scenario is composed of. From the requirements of the use cases, the *RECAP optimisation framework* is planned to have multiple VM admission and resource consolidation policies available. Some of the policies typically also have additional configuration parameters required for execution. The *Experiment Manager* will provide an overlay mechanism that queries the *RECAP optimisation framework* in order to validate its configuration parameters. This connection will improve the usability of the system, ensuring the minimisation of errors prior to running a simulation experiment.

Different model generation routines will be available to generate system configurations beyond the scale of prototypical testbed systems using historical data logs allowing for the creation of simulated testcases for large scale system at possible, but rarely experienced corner cases. The *Experiment Manager* provides an API that allows for the querying of different model generators for their properties and types of models that they can generate. The simulation models can also be built using the RECAP data collection framework. Current system infrastructure state can be derived directly from the real-time monitoring information and other aspects of system behaviour are modelled using historical data analysis. Experiment Manager provides API calls and methods to interface with the data collection framework and retrieve simulation ready artefacts based on the real data.

Finally, the *Experiment Manager* provides an API to start, stop and query the simulation run using the *Simulation Manager* component. Once all of the necessary models are populated, a simulation experiment is passed to the *Simulation Manager* and saved in the experiment database for further

reference. A unique experiment run ID is generated, and can be used to query the simulation status and to collect simulation results.

Simulation Manager *(related requirements: R701 R707, R717)*

The *Simulation Manager* is responsible for spawning simulation processes. Each simulation process runs as an independent entity, allowing for multiple simulations to take place in parallel. The *Simulation Manager* also keeps track of all running processes and computing resources that are utilised. Future developments will also focus on distributing simulations over multiple hosts for more efficient resource allocation. By running simulations in parallel, one can execute multiple experiment variations at the same time. For example, by using the same system models but changing optimisation policies the effects can be obtained simultaneously, hence saving time comparing to sequential executions of experiments. In this way the RECAP simulation platform can provide better control over the model variety and results correlation.

Event Generator *(related requirements: R701, R707, R708, R709, R715, R716)*

Every action within the simulation framework has to be translated into a simulation event that is either created during the simulation experiment run or scheduled to be executed at a certain time. The *Event Generator* component is responsible for creating events of certain types to fit the simulation time duration. In the current design of the *RECAP simulation framework* there are two event types supported: new VM arrival events, and periodic optimisation events. As the name suggests, new VM arrival or departure events represent a creation or decommissioning of a VM within the system which can occur when new fog application is being deployed, when an existing application needs to be horizontally scaled or customer removes services from fog.

Model Mapper *(related requirements: R701, R702, R703, R704, R705, R706, R707, R708, R709, R710, R711, R712, R713, R714, R715, R716, R717, R718)*

At a higher level of RECAP, the edge system models are defined to suit the description of use-cases (see Chapter 4.2). The RECAP system models are used across data collection, optimisation and simulation toolkits to ease integration. However, these models differ from the models that are needed for various cloud simulation engines that can be used to extend the RECAP simulation platform. The Model Mapper component ensures the mapping of the RECAP model components onto simulation engine model components.

Event Coordinator *(related requirements: R701, R707, R708, R709, R710, R714, R716)*

The *Event Coordinator* is an auxiliary component that liaises between the *Simulation Engine*, *Model Mapper* and the *Optimisation Framework* (see Figure 3). The *Event Coordinator* contains methods to start the *Simulation Engine* after obtaining appropriate models from the *Model Mapper*. The *Event Coordinator* also has two call-back methods for periodic events and new VM arrivals to be triggered during a simulation process execution. Depending on the type of injected events, a different optimisation action is called from the *Optimisation Framework* and later applied through the event coordinator by updating both models. Due to the tight coupling with the *Simulation Engine*, the *Event Coordinator* is also responsible for serialising simulation results to database storage and for the coordination of optimisation and failure events.

Simulation Engine *(related requirements: R701, R702, R703, R704, R705, R706, R707, R708, R709, R710, R711, R712, R713, R714, R715, R716, R717, R718)*

Once the system model and experiment parameters are complete, the simulation process can be started. The *Simulation Engine* is an abstract name for a simulation framework capable of calculating system parameters as the time progresses. The main part of a DES engine is an Event Queue. An Event Queue represents a simulated experiment time duration, with different events being triggered at specific points in time. For DES, the time frame is not related to the wall clock time as it is a simulation of the time progression. Depending on the number of occurring events, the difficulty/complexity of calculations, and the hardware upon which the simulation is being executed, one second of simulation time might be faster or slower than the wall-clock time.

The source consumption is calculated by firstly defining a particular resource processing capacity and secondly inducing workload to that resource. For example, a CPU is a resource capable of processing a certain amount of instructions per second, and each user action within the application costs a certain amount of instructions to be performed. To calculate how long, it will take for a CPU to process a user request (task) the capacity of the CPU is divided by the request demand. In the virtualised setup one CPU can be shared amongst multiple applications and it is up to hypervisor to regulate access to the resource through the use of specific provisioning policies. Cloud simulation frameworks also account for such system behaviour by supporting different provisioning policies for task execution.

4.2 Implementation

The implementation of the RECAP simulation framework was written using the Java programming language. Java language was chosen because of its popularity, cross-platform portability and available expertise within our research group, the RECAP consortium and wider research community. As shown in Figure 3 and later in Figure 4, the RECAP simulation framework is designed with the aim of meeting the RECAP project objectives by building on top of the existing proven simulation solutions. Such an approach allows for later seamless integration with the *Data Collection* and *Optimisation* frameworks, as well as the ability to swap between simulation frameworks depending on experimental requirements in terms of execution speeds and required granularity levels of experimental models.

The work done for the initial RECAP simulation framework implementation focused mainly on the mechanics of the software component interactions, data flow and model transformations. The concept of *Experiment Scenario* was introduced to gather all models, events and configuration data into one object which can then be checked for errors within the *Simulation Mapper* and then passed on further to the *Simulation Manager*. The *Simulation Manager* can then map RECAP models to models of a cloud simulation framework which run on the *Simulation Engine* and start a simulation process. The component diagram shown in Figure 4 shows implementation states of designed components. This includes components completed for the initial version (which will be further developed in future versions), partially implemented components as well as future components.

In the initial simulation platform release, we have integrated CloudSim (Calheiros, Ranjan, Beloglazov, De Rose, & Buyya, 2011) as a simulation engine and have written a model mapper class that holds object references between the *RECAP Infrastructure Model* (RIM) and CloudSim models (A detailed description of the RIM is presented in (D8.1 Initial infrastructure modelling and resource mapping, 2017)).

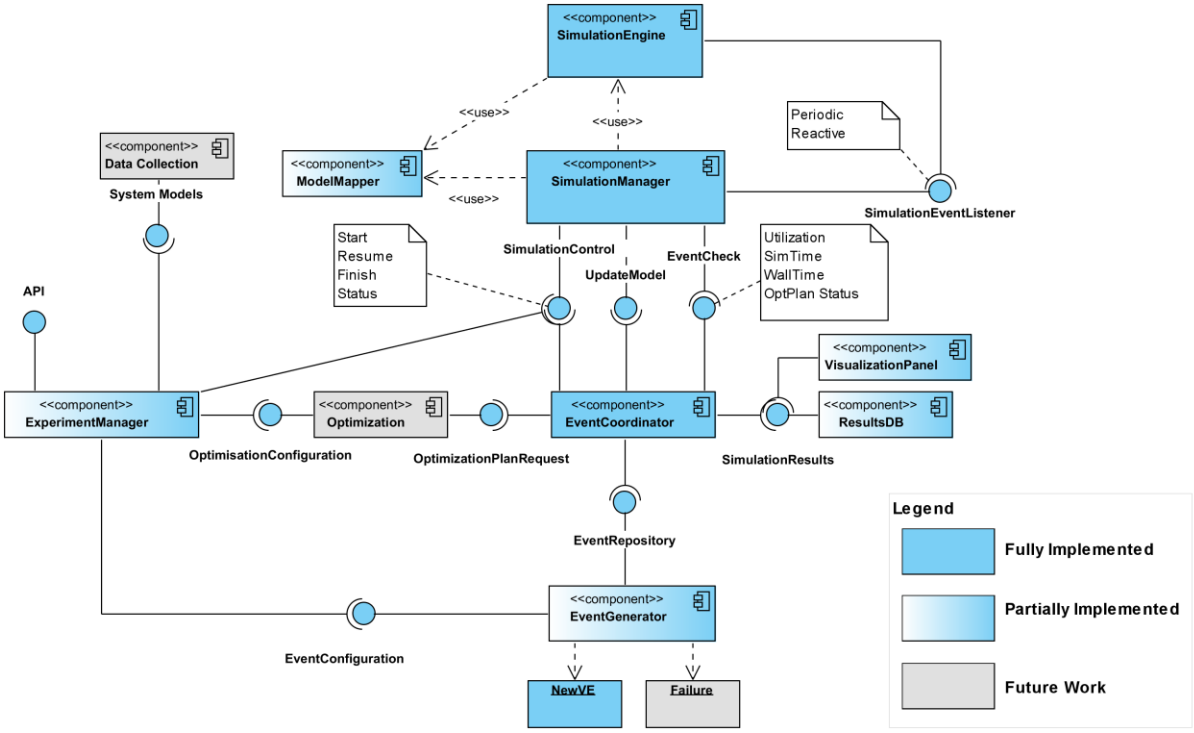


Figure 4: RECAP simulation framework implementation component diagram

The relations between a RIM and a CloudSim infrastructure model are captured inside the current version of Model Mapper. Even though work on creating RECAP-specific model formats and experiment scenarios is still ongoing, the initial simulation framework can use some of the modelling concepts directly from CloudSim simulator.

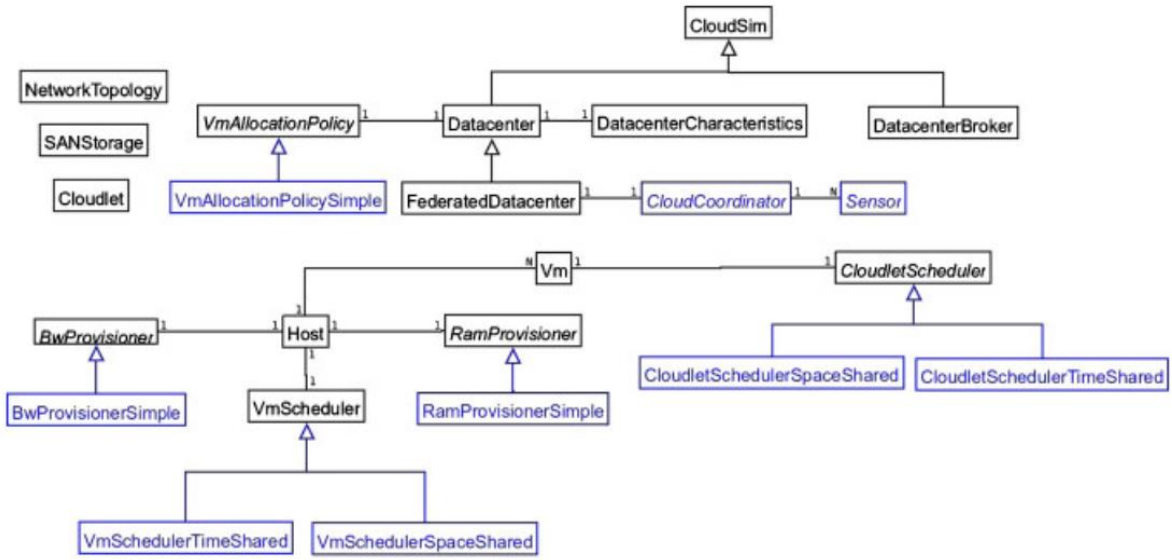


Figure 5: CloudSim model class diagram (Calheiros, Ranjan, Beloglazov, De Rose, & Buyya, 2011)

The CloudSim core class model, shown in Figure 5, consists of following components:

Datacenter depicts a collection of hardware hosts assigned through the *DatacentreCharacteristics* class. Each *Datacenter* has *VMAllocationPolicy* class that is responsible for VM allocations to

hardware resources within data centre and *DatacenterBroker* class which is responsible for choosing datacentre according to the QoS of the application.

Host represents the physical hardware that is inside the datacentre node. It contains the description of compute resources such as CPU, Memory and Storage. As shown in Figure 5, Host component has resource allocation policies defined for each type of resource, i.e., *BwProvisioner*, *RamProvisioner*, *VmScheduler* these policies are algorithms which are used for estimating hardware resource sharing between multiple VMs on a single virtualised host in the datacentre.

VM similarly to the Host class, the Vm class has a definition of resources it requires to run on in terms of CPU, Memory and Storage. In addition, each Vm has a *CloudletScheduler* class instantiated, as the name suggests, it provides algorithms for distributing VM resources to multiple user request (cloudlet) processing.

Cloudlet class models represents an atomic expression of application resource request. The base implementation has a pre-assigned instruction length, but can be extended to model more complex application behaviour.

In order for CloudSim to be integrated within the RECAP simulation platform, RECAP system models that are used by Data Collection and Optimisation framework require to be mapped onto CloudSim models shown in Figure 5. Also, the core functionalities of *DatacenterBroker* and *VmAllocationPolicy* need to be extended to satisfy project specific domain and use-case requirements. In the case of RECAP, the VM allocation decisions are made solely by the optimisation component propagating to other requirements (i.e. R701, R705, R709, R711). Therefore, each decision as to which data centre to choose and which node to choose is executed via sending requests to optimisation component via integration modules.

Infrastructure model mapping

There is a requirement for a shared infrastructure model structure between the simulation data collection and optimisation components (WP702, WP703, WP711). By having the same model format, all three components can exchange system descriptions natively as long as the model format is adhered to by all parties, thereby reducing the integration effort. Hence, the *RECAP Infrastructure Model* (RIM) was developed in order to capture the virtualised hardware configuration and topology of fog/edge/cloud based systems.

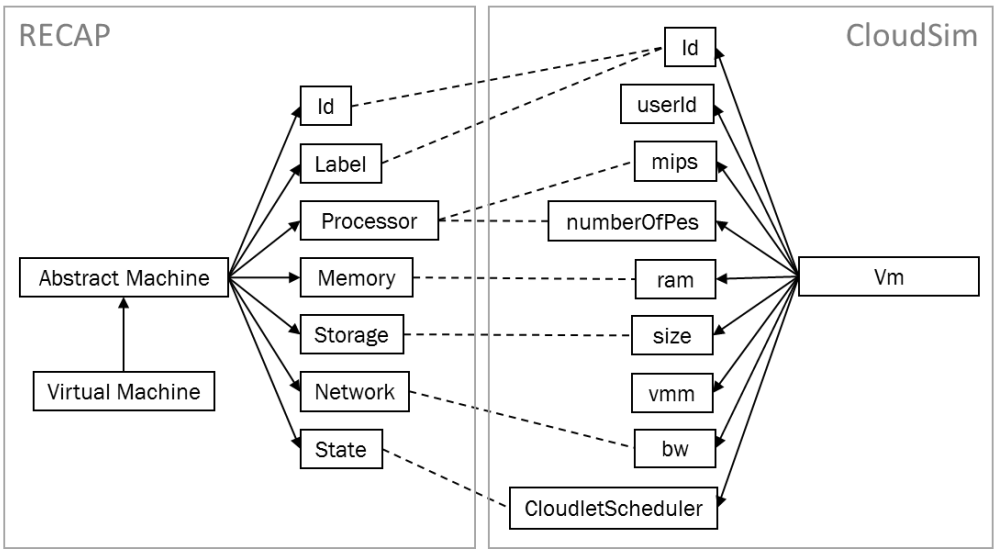


Figure 6: RECAP Virtual Machine class and CloudSim Vm class model relations diagram

The Abstract Machine element shown in Figure 6 and Figure 7 represents a collection of fundamental computing resources, i.e., Processor, Memory, Storage and Network which can be used for describing configuration of both physical and virtual machines. The "`State`" enum attribute represents the current operational state of the entity which can be set to "`ON`", "`OFF`" or "`SUSPENDED`". Other attributes are the "`ID`" and "`Label`", where ID serves as a unique identifier of the element and the label is thought to serve as more human readable name for the entity.

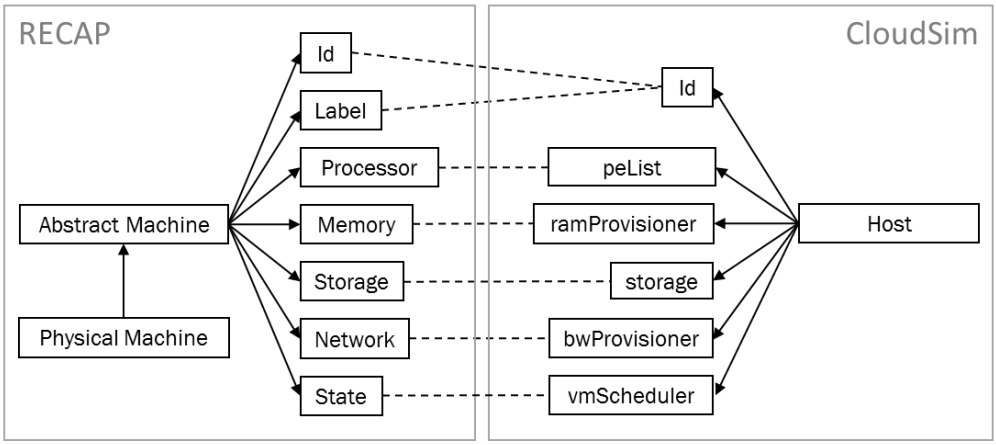


Figure 7: RECAP Physical Machine class and CloudSim Host class model relations diagram

The RIM captures Processor as an array of CPUs where each CPU also has an array of cores. Each core has a defined minimum and maximum frequency range and the current frequency value at which it operates. These attributes allow for the capturing of the current state of each CPU core independently. The CloudSim virtual machine model of class "`Vm`" captures VM configuration as number of the processing elements "`numberOfPes`" and "`mips`", these values refer to number of cores and the processing capacity of each core in Million Instructions Per Second (MIPS). The physical compute node depiction in CloudSim consists of list of Processing Entities (PE) expressed as "`peList`" object. Each PE has an id, capacity in MIPS and a "`PeProvisioner`" attributes assigned. The latter represents the type of algorithm of sharing processing capacity between multiple VMs. Mapping of RECAP processor model core is done directly onto the each PE. The PE provisioning algorithm is derived from the hypervisor CPU scheduler characteristics.

The memory component, shown in Figure 6 and Figure 7, is also part of Physical Machine (PM) and Virtual Machine (VM) configuration resources. The memory is modelled as a single block unit with the current capacity and available capacity values. In the case of the PM, the memory object represents the total memory located on the node in the data centre. In this case, the memory of a VM represents the share of physical memory that was assigned to it. In the CloudSim "`Vm`" model class memory is expressed as "`ram`" attribute type integer. Which is mapped to a total memory capacity within RECAP model. In case the CloudSim "`Host`" class model the "`RamProvisioner`" type of class is defined that holds total amount of "`ram`" and the memory provisioning policies of a hypervisor that provides resources to deployed VMs.

The RECAP Storage component contains the logical array of storage devices. Each storage device has the available capacity and current capacity value properties, but in addition to that it also has additional attributes of performance and type. The performance value refers to the speed of data transfer and type refers to the pre-defined enum types of "`HDD`", "`SSD`" and "`REMOTE`". Defined types refer to the type of storage (i.e. Hard Disk Drive (HDD), Solid State Drive (SSD), Network Access Storage (NAS)) that is assigned for PM or VM functions. In case of CloudSim model "`Vm`" model storage captured as "`size`" type long variable, therefor, RECAP storage model total storage capacity is calculated and mapped to the "`size`" value. The CloudSim "`Host`" model also contains only a long type variable named "storage" holding the total amount of storage capacity available for the compute node.

The RECAP network model consists of a Network component containing an array of Network Interface Cards (NIC). Each NIC holds an array of string addresses and current and available capacity values representing the bandwidth. The network model is still under development and will be expanded to address inter-site communications further into the project. In case of CloudSim VM model it only defines a bandwidth assigned to a VM by the "`bw`" type long attribute value. In case of "`Host`" model the class type "`BwProvisioner`" is defined. As the name suggests it is responsible of allocation of compute node network bandwidth to the VM entities.

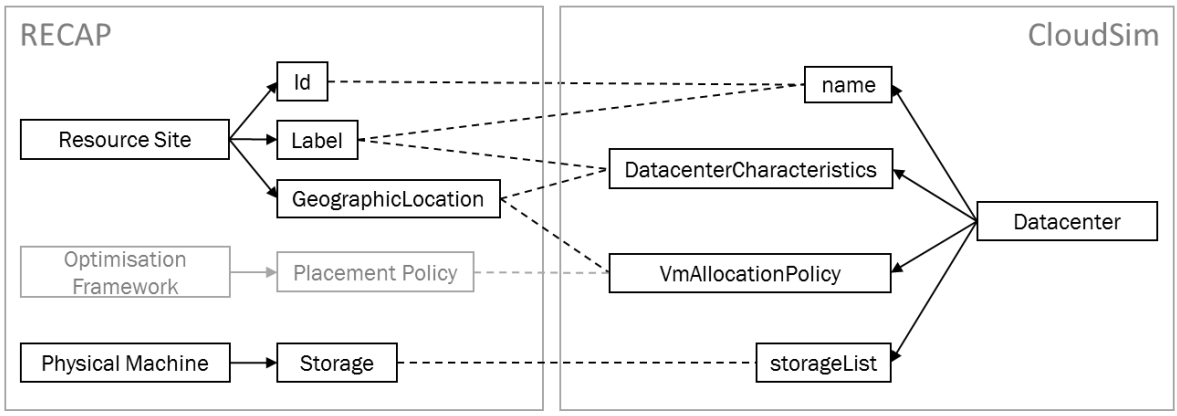


Figure 8: RECAP Resource Site class and CloudSim Datacenter class model relations diagram

In the RECAP infrastructure model each Physical Machine model element has a Resource Site component assigned which represents its physical location. Resource site groups physical hardware into geographically distributed datacentres that can serve as edge, core or cloud locations. Each such location has attributes of latitude and longitude, ID and Name. The core principles of fog computing (Iorga, et al., 2017) and requirements derived from use-cases (R711) depends on a model that is able to support distributed topology of resources that are spread geographically due to the effects user proximity has on latency and cost of services. The CloudSim

simulator model has a similar notion of distributed resources expressed within "`Datacenter`" class, Figure 8 shows mapping between the RECAP components and the CloudSim components. The geographic location coordinates are not used directly in CloudSim, but the location can have an impact on the parameters of "`DatacentreCharacteristics`" e.g. cost, time zone. The class "`VmAllocationPolicy`" provides an algorithm of admitting VMs to compute nodes, in RECAP project VM allocation decisions will be made in future releases through the integration with the optimisation framework.

Both RECAP infrastructure models and CloudSim models are essentially describing the same hardware system, which makes model attributes the same or similar in nature. It is important to translate elements described within RECAP model into the CloudSim model programmatically. In Figure 9 the implementation between model translations is shown for method *createHostListSim()* written in Java programming language. When this method is called, a for loop is created (line 4) which iterates through all the Physical Machines in the RECAP model and creates a list of Hosts within CloudSim model.

```
1.  private void createHostListSim() {
2.      CloudSimMapper.hostListSim = new ArrayList < PowerHost > ();
3.      int nodeId = 0;
4.      for (PhysicalMachine node: Experiment.getPmList()) {
5.
6.          List < Pe > peList = new ArrayList < Pe > ();
7.
8.          if (CloudSimMapper.NodeCorrespondance.size() != 0) {
9.              nodeId = CloudSimMapper.NodeCorrespondance.lastKey() + 1;
10.         }
11.
12.         for (int i = 0; i < node.getProcessor().getNrCPUs(); i++) {
13.
14.             for (int n = 0; n < node.getProcessor().getCPUs()[i].getNrCores(); n++) {
15.
16.                 // each core is a processing entity PE
17.                 Core core = node.getProcessor().getCPUs()[i].getCores()[n];
18.
19.                 int mips = core.getFrequency();
20.                 Pe pe = new Pe(peList.size(), new PeProvisionerSimple(mips));
21.                 peList.add(pe);
22.             }
23.
24.         }
25.
26.         int ram = node.getMemory().getCapacity() * 1000; // host memory (MB)
27.         long storage = node.getStorage().getCapacity() * 1000; // host
28.
29.         // storage
30.         int bw = node.getNetwork().getCapacity(); // Megabits/s
31.
32.         // vars for linear power model
33.         double maxPower = 800;
34.         double staticPowerPercent = 100;
35.
36.         PowerModel powerModel = new PowerModelLinear(maxPower, staticPowerPercent);
37.
38.         CloudSimMapper.hostListSim.add(
39.             new PowerHostUtilizationHistory(
40.                 nodeId,
41.                 new RamProvisionerSimple(ram),
42.                 new BwProvisionerSimple(bw),
43.                 storage,
44.                 peList,
45.                 new VmSchedulerTimeSharedOverSubscription(peList),
46.                 powerModel
47.             )); // This is our machine
48.
49.         // update our node correspondence
50.         CloudSimMapper.NodeCorrespondance.put(nodeId, node.getId().toString());
51.
52.     }
53.
54. } // end of create host list
```

Figure 9: RECAP PhysicalMachine class mapping to CloudSim Host class

The number of Processing Entities (Pe) in the CloudSim model is mapped to the number of CPU cores within the RECAP model, their frequency translated directly to Millions Instructions value. The Memory, Network and Storage capacity values from the RECAP model mapped directly to integer values of corresponding elements within CloudSim model. Modelling of Power consumption is still in development therefore the power model is maintained with default values.

```
1.  public List < PowerVm > convertToCloudSimVmList(List < VirtualMachine > rimVmList, int total
    NumberOfCloudlets) {
2.
3.   List < PowerVm > cloudSimVmList = new ArrayList < PowerVm > ();
4.
5.   for (VirtualMachine vmRim: rimVmList) {
6.    // VM description
7.    int vmid = (int) System.currentTimeMillis();
8.    this.getVmCorrespondance().put(vmid, vmRim.getId().toString());
9.    int mips = vmRim.getProcessor().getMaxFrequency();
10.   // no data in the RIM model yet
11.   long size = 10000; // image size (MB)
12.   int ram = vmRim.getMemory().getCapacity() * 1000; // vm memory (MB)
13.   // RIM BW is in Mbit/s
14.   long bw = vmRim.getNetwork().getCapacity();
15.   int pesNumber = vmRim.getProcessor().getNrCores(); // number of cpus
16.   // no data here in RIM
17.   String vmm = "Xen"; // VMM name
18.   // create VM
19.   int vmPriority = 1;
20.   // not really sure what this does
21.   double schedulingInterval = 0.0;
22.   // this scheduler very important!!
23.   PowerVm vmSim = new PowerVm(
24.    vmid, this.dcBroker.getId(),
25.    mips, pesNumber, ram, bw, size, vmPriority, vmm,
26.    new CloudletSchedulerDynamicWorkload(mips, pesNumber), schedulingInterval);
27.   this.createCloudletList(totalNumberOfCloudlets, dcBroker.getId(), vmSim.getId());
28.   // add the VM to the CloudSim VMList
29.   cloudSimVmList.add(vmSim);
30.  }
31.  return cloudSimVmList;
32. }
```

Figure 10: RECAP VM to CloudSim VM mapping

The mapping of the VMs between RECAP and CloudSim models is done in similar manner using method called *convertToCloudSimVmList()*. As shown in this method accepts list of RECAP VMs and an integer value of cloudlet value. The method creates a loop (line 5) that traverses through all of the RECAP VMs and creates an equivalent VMs for the using CloudSim models. The capacity of virtual Memory, Storage and Network from RECAP model are mapped accordingly to the Memory (RAM), Volume size and Bandwidth in the CloudSim model. The virtual CPU configuration is mapping CPU speed and cores to Millions Instructions per virtual Processing Entity (PE).

Application Model

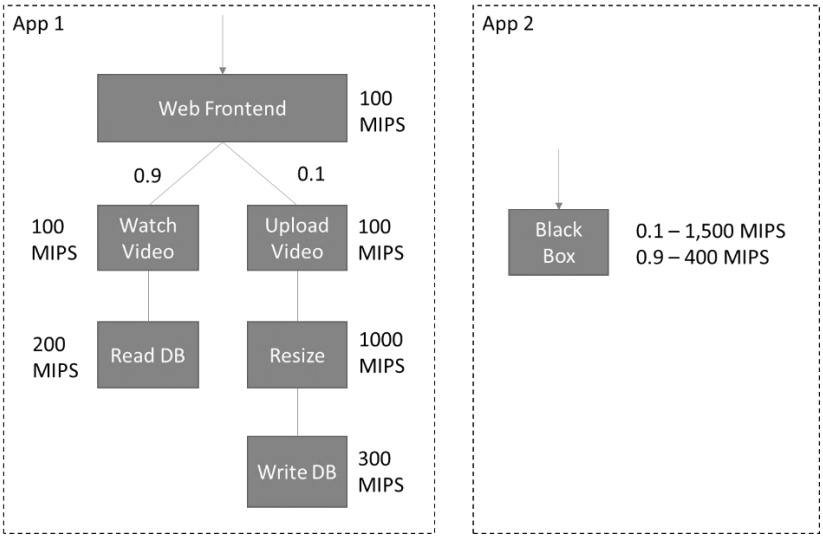


Figure 11: Application model example

As the name suggests the application model describes the layout, links and resource demands of the application components. The model itself can be as complex as direct blueprint of application class diagram or very abstract represented by a single probability function.

The simplified example in Figure 11 shows two ways of modelling the same application. On the left "App1" represents a general design of a video sharing website where user can watch and upload videos. The application model shows that there is 90% chance that an arriving user watches the video, and 10% chance that the user uploads it. If users choose to watch a video they have to access the web frontend and select the video, which is then fetched from the data base. In the example, each part of the application requires a certain amount of CPU MIPS to run. When the video is just being watched, it consumes in total 400 MIPS, but if one chooses to upload video content, it would require significantly more CPU resources due to video resizing and formatting, and amounts to 1,500 MIPS.

In the "App2" example the same application is modelled as a black box component with probability of 90% to consume 400 MIPS and 10% to consume 1,500 MIPS. A higher abstraction level usually yields less accurate results in simulation, but still can be useful if, e.g., only limited information about the application is available.

In order to derive resource requirements for the application it needs to be tested (benchmarked) in a controlled environment. During the tests, data about hardware resource utilisation is collected and correlated back to the application launch parameters. It is also possible to build an application model from a large historical data set collected over time from the system, however this process might not exhaust all of the application behaviour traits.

Cloudlet

```
public Cloudlet(int cloudletId,
        long cloudletLength,
        int pesNumber,
        long cloudletFileSize,
        long cloudletOutputSize,
        UtilizationModel utilizationModelCpu,
        UtilizationModel utilizationModelRam,
        UtilizationModel utilizationModelBw)
```

Allocates a new Cloudlet object. The Cloudlet length, input and output file sizes should be greater than or equal to 1. By default this constructor sets the history of this object.

Parameters:

`cloudletId` - the unique ID of this Cloudlet

`cloudletLength` - the length or size (in MI) of this cloudlet to be executed in a PowerDatacenter

`cloudletFileSize` - the file size (in byte) of this cloudlet `BEFORE` submitting to a PowerDatacenter

`cloudletOutputSize` - the file size (in byte) of this cloudlet `AFTER` finish executing by a PowerDatacenter

`pesNumber` - the pes number

`utilizationModelCpu` - the utilization model cpu

`utilizationModelRam` - the utilization model ram

`utilizationModelBw` - the utilization model bw

Figure 12: Cloudlet constructor detail[2]

The CloudSim core application model is represented in a form of cloudlet (`org.cloudbus.cloudsim.Cloudlet` class). The cloudlet represents a unit of work that is sent to the system for processing. As shown in Figure 12 the class constructor includes parameters that describe cloudlet processing resource demand which are: Millions of Instructions, number of CPU cores referred to as processing units (`pesNumber`) and file input and output sizes. Each cloudlet also is assigned a utilisation model for CPU, memory, and network bandwidth that governs the access to resource usage.

[2] Source: http://www.cloudbus.org/cloudsim/doc/api/org/cloudbus/cloudsim/Cloudlet.html

```
1.  private void createCloudletList(
2.      int numberOfCloudletsPerVM,
3.      long length,
4.      long fileSize,
5.      long outputSize,
6.      UtilizationModel utilizationModel,
7.      int brokerID,
8.      int vmID) {
9.
10.     int counter = 0;
11.     while (numberOfCloudletsPerVM > counter) {
12.
13.         Cloudlet cloudlet = new Cloudlet(
14.             counter,
15.             length,
16.             1,
17.             fileSize,
18.             outputSize,
19.             new UtilizationModelStochastic(),
20.             utilizationModel,
21.             utilizationModel
22.         );
23.
24.         cloudlet.setUserId(brokerID);
25.         cloudlet.setVmId(vmID);
26.         this.cloudletList.add(cloudlet);
27.
28.         counter++;
29.     }
30. }
```

Figure 13: RECAP Simulation Cloudlet generation code

In the RECAP initial simulation platform version, the application model implementation is based on creating a list of cloudlets per individual VM. The Java method shown in Figure 13 accepts standard cloudlet parameters plus the number of cloudlets per VM, broker ID and VM ID to which these cloudlets will be submitted during the course of simulation experiment. This is done for each VM that is submitted to the system.

In the NetworkCloudSim (Garg & Buyya, 2011) extension, the AppCloudlet class was created (org.cloudbus.cloudsim.network.datacenter.AppCloudlet). The AppCloudlet contains NetworkCloudlets list that are able to represent more complex application structure of multiple stages. Such functionality suits better RECAPs functional requirements, therefore in future releases the functionality of AppCloudlet class will be adopted.

Workload Model

The workload model captures user interaction with the system. It can be defined as arrival rate, i.e., number of user requests per second, or as an actual request number at points of simulation time. In some cases, depending on the simulation platform design, definitions of user requests can partially or fully take over the application models. For example, a user request entity itself can represent a “job” with certain parameters of resource demands attached. The job is then submitted to a hardware node for processing.

```
1.  protected void submitCloudlets() {
2.          int vmIndex = 0;
3.          List < Cloudlet > successfullySubmitted = new ArrayList < Cloudlet > ();

4.          for (Cloudlet cloudlet: getCloudletList()) {
5.              Vm vm;
6.              // if user didn't bind this cloudlet and it has not been executed ye
    t
7.              if (cloudlet.getVmId() == -1) {
8.                  vm = getVmsCreatedList().get(vmIndex);
9.              } else { // submit to the specific vm
10.                 vm = VmList.getById(getVmsCreatedList(), cloudlet.getVmId());
11.                 }
12.             }
```

Figure 14: Extract from the CloudSim DatacentreBroker class

A cloudlet is an expression of unit of work that is sent by user to the system for processing. Each cloudlet requires to gain access to virtualised hardware resources to be completed. In the case of CloudSim cloudlets are submitted to VMs by a *DatacentreBroker* class using *submitCloudlets()* method. If a cloudlet was not assigned to a particular VM at a creation time, a suitable VM will get picked by a broker. In the initial version of RECAP simulation platform a list of cloudlets is assigned to a particular VM using *setVmId(int vmID)* method as shown in Figure 13 line 25.

5 Preliminary evaluation

To demonstrate the initial simulation framework capabilities, a simulation experiment was designed and executed based on the infrastructure topology of one of the use cases. The purpose of this exercise is to traverse through the major steps in the simulation experiment process, i.e., system modelling, simulation and result analysis. In addition, while running the experiment, the simulation framework performance is of interest in terms of system resource consumption for running the experiment. The RECAP simulation framework and CloudSim integrated cloud simulation framework both are developed in Java. Therefore, in order to capture resource consumption, the Java Virtual Machine (JVM) activity is monitored within the duration of the experiment.

The experiment setup begins with the creation of a system model. As a basis for the infrastructure model, we take the UK-wide BT site locations. The model was selected as it represents a widespread domain challenge in a complex and realistic geographical setting. As shown in Figure 15, pins on the map represent a physical location of multiple, geographically distributed sites, where hardware resources are deployed and interconnect by a network. The BT UK infrastructure topology consists of 1,131 site locations marked by yellow markers, which is a mix of core and

metro nodes. The markers are plotted on the map using the Site information captured using the RECAP Infrastructure Models (RIM) as described in Chapter 4.2.

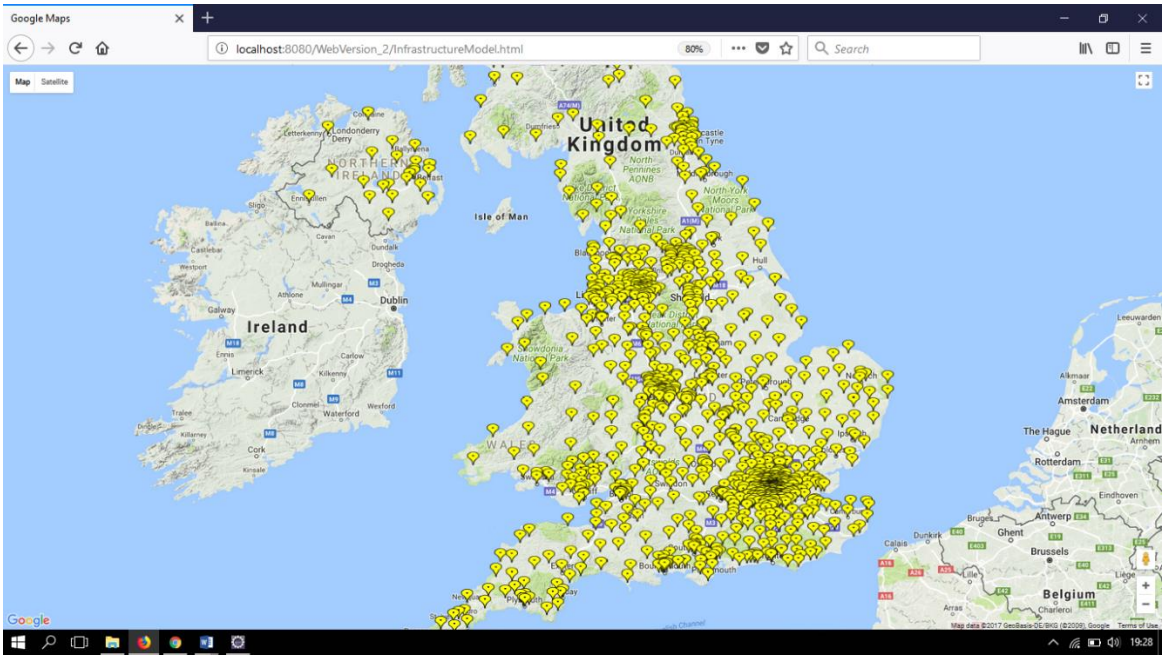


Figure 15: BT UK site locations

The visual analysis of the site locations illustrates the correlation between population density and number of sites. In the Figure 16 we can see that the large cities such as London, Birmingham and Manchester have larger site density to cater for the greater population service demands.

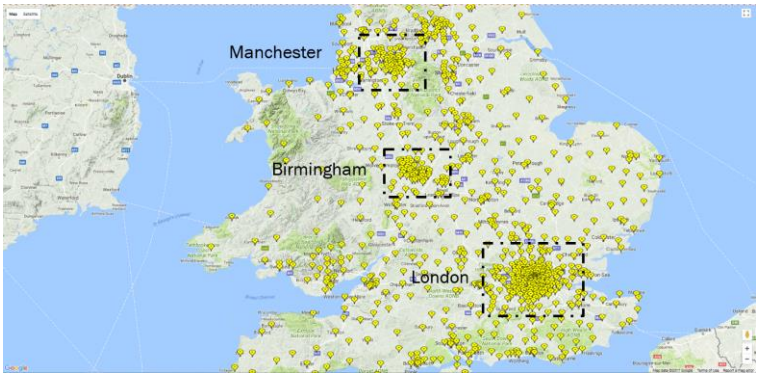


Figure 16: BT UK site locations clusters

While it is yet unknown what the exact hardware deployed at each site location will be, for the sake of this experiment we assume that each of 1,131 sites has 10 hardware nodes deployed. Each node is modelled to have 4 processing units with 2400 MIPS capacity, 16GB of memory and 1TB of storage. Virtual system configuration consists of 1000 VMs each provisioned with 2 CPU cores of 1000 MIPS per core, 0.5GB of memory and 1GB of image size. To induce workload on the system we admit 50 cloudlets per minute of 50,000 MIPS for each cloudlet. Simulation experiment duration is set to simulate 60 minutes of BT infrastructure model based system. A summary of the system model is provided in Table 4.

Table 4: Simulation experiment system model summary

Total Sites	**1,131**
Total Nodes	11,310
Total VMs	22,620
Total Cloudlets	3,000

After simulation experiment execution, the results are currently generated using the CloudReports plugin (Sá, Calheiros, & Gomes, 2014). The extracts from the simulation report feature overall CPU consumption and overall memory consumption shown in Figure 17 and Figure 18 respectively.

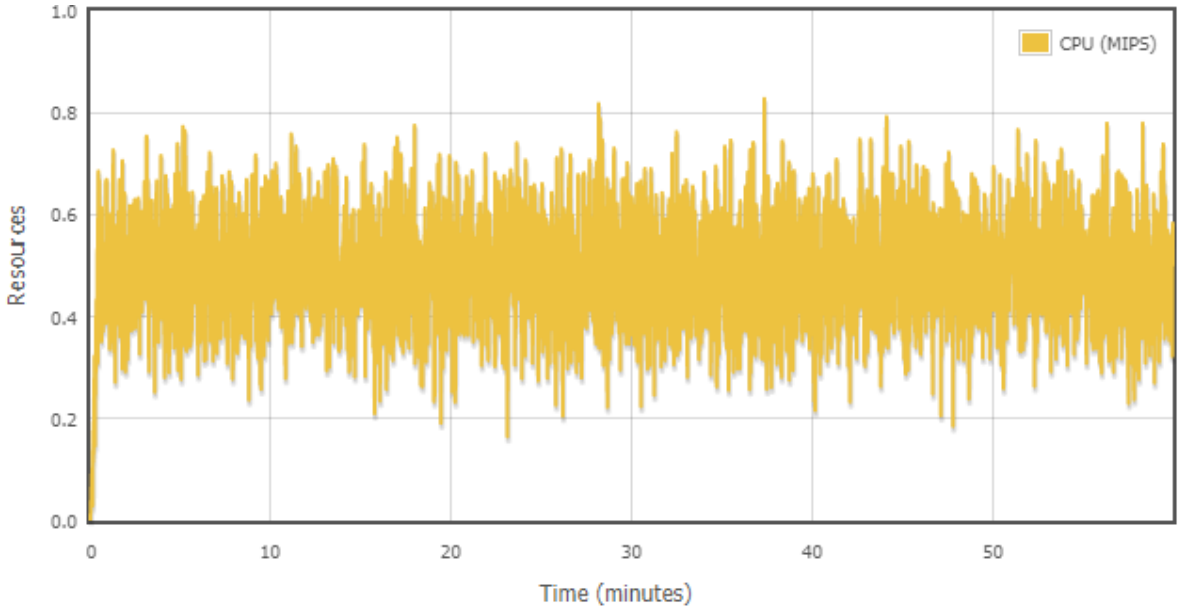


Figure 17: Overall simulated CPU consumption

The overall CPU consumption shows the demand of processing capacity over 60 min of simulation duration. From the result (cf. Figure 17) we can see that the combined processing power of all 45,240 CPU cores available in the system is utilised mostly in the region of 40%-50%, with the maximum being above 80% and minimum below 20%, after the initial 1-2 minute warm-up phase. Since our model does not include geographic user distribution the workload is being distributed equally over all of the nodes on all sites.

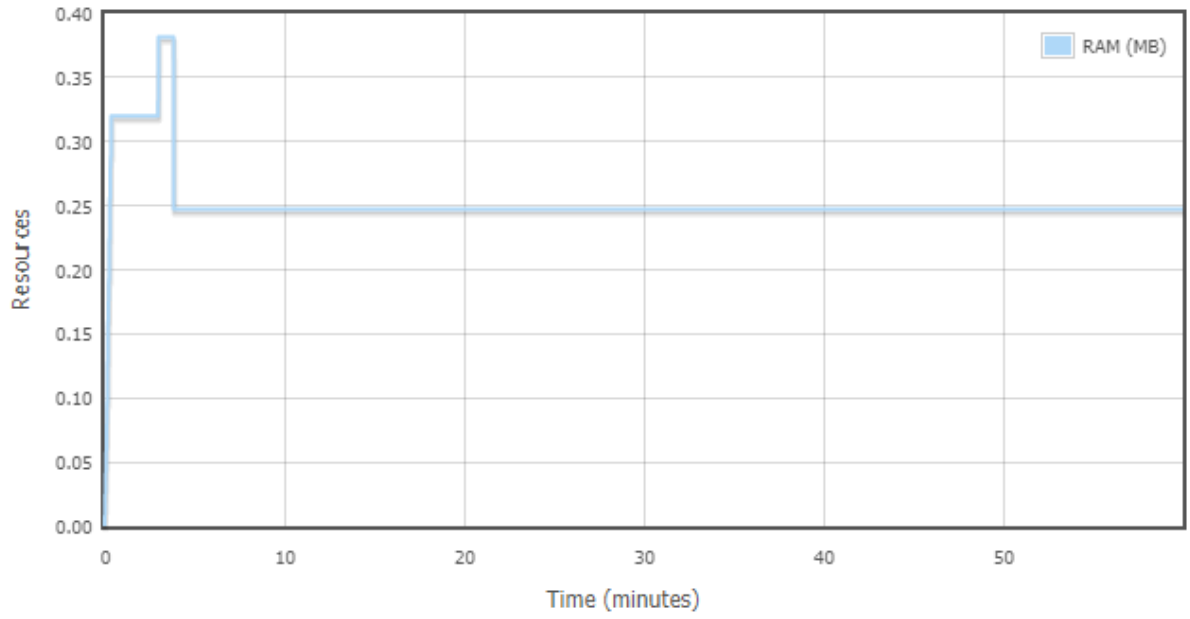


Figure 18: Overall simulated memory consumption

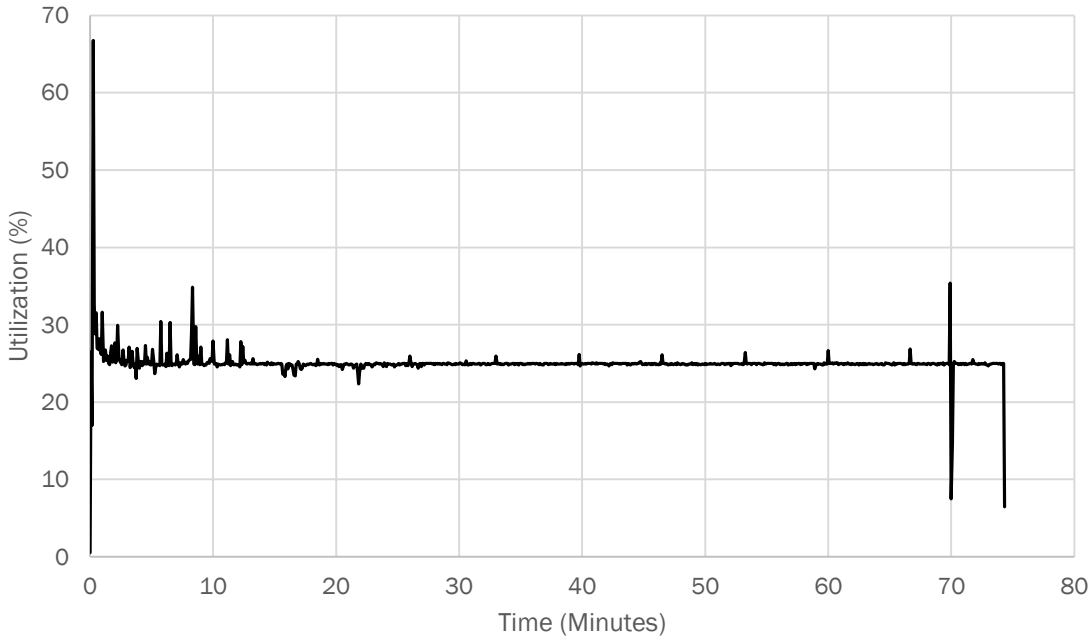
From the overall memory consumption report (Figure 18) it can be seen that the initial memory consumption spike reaching above 35% of total 180.96 TB due to VM admission to the system which then normalizes at 25% once the VM admission is completed.

Simulation results also include individual node resource consumption, cost and power estimations for system overall as well as for individual nodes. These metrics were out of scope of current experiment and planned for the later releases once the project progresses.

Above presented simulation results demonstrate overall system resource capacity utilisation calculated according to the modelled user workload. Based on these results use case owners can predict resource demand needed to serve required system user needs. Future integration with the RECAP optimisation framework will also allow to gauge the resource distribution including influence of autonomic optimisation decisions.

5.1 Simulation framework performance

The simulation experiment described in Section 5 was executed on a PC machine with the configuration of Intel i7-6600U CPU @3.3 GHz with 2 cores and 4 threads plus 16 GB of DDR4 memory. During the simulation experiment the Java process demand was monitored for CPU and memory using Psutil pyton library.



.

Figure 19: Simulation framework CPU utilization

The CPU utilization, shown in Figure 19, remained largely at around 25% among all the cores with the initial spikes in the beginning during the warmup phase and the end at the simulation result generation phase. The memory utilization graph, shown in Figure 20, shows gradual memory demand increase throughout the experiment with the jump at the beginning of simulation at the warmup phase and at the end of simulation at the report generation phase.

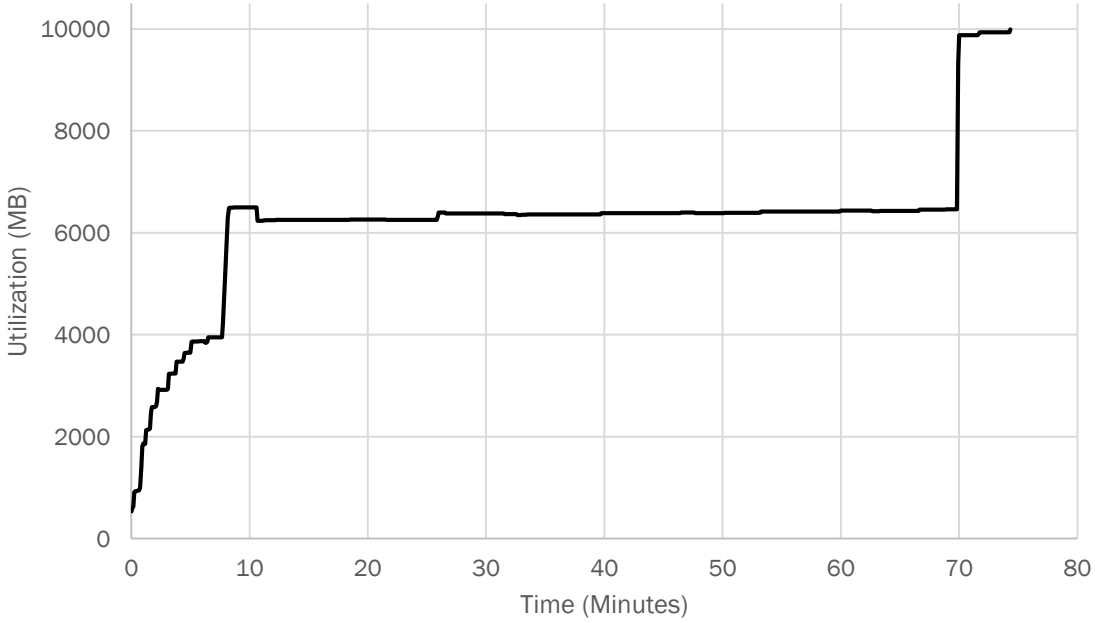


Figure 20: Simulation framework memory utilization

These are the first series of RECAP simulation framework tests which, as expected, show high memory demand and low CPU utilisation rate. The large memory demand comes from the size of

the simulated system model. Each element in the model and each occurring event is a Java class object that requires its own memory space, and since the model remained the same throughout the simulation experiment (in Figure 20) a straight line can be seen from the 10th to the 70th minute. Usually memory can be freed by removing no longer used and inactive Java objects with such operation being performed automatically during the runtime by the Java Garbage Collector, however since all of the simulation objects are expected to be used the memory cannot be released until the simulation experiment execution is completed.

Comparing to the memory demand, the CPU is largely underutilised (see Figure 19), with such behaviour occurring due to sequential processing of events within the queue. The simulation events are arranged strictly by time. Due to occurring interdependencies within simulation models, in this initial version simulation events cannot be executed before their time. Such a central queue can be a speed bottleneck for the existing DES frameworks when simulating large scale models, and these test results confirm that this is the case. In the next phases of the project, potential solutions for this will be explored.

6 Conclusions

The supporting documentation demonstrates the reasoning behind the simulation framework requirements elicitation process. The list of functional requirements for the RECAP initial simulation framework is presented in Table 3, being based on the direct communication with the use-case partners, the use case description document D3.1 (D3.1 Initial Requirements, 2017) and the review of available open source simulation platforms (published in (Lynn, et al., 2017) and (Byrne, et al., 2017)).

Based on these derived requirements the initial simulation platform component and data flow designs have been created. The current version of RECAP simulation framework focuses on a complete end-to-end simulation process from the edge/fog/cloud system model creation, simulation execution and finally results delivery. The existing cloud simulation platform CloudSim was integrated with the available RECAP infrastructure models and the overall component functionality. This exercise showed the similarities between the available CloudSim modelling functionalities, such as the infrastructure model that can be re-used, but also highlighted the areas that needs to be extended to fit RECAP use case objectives, such as application, workload and failure modelling.

The next steps in this work package in terms of developing the RECAP simulation platform will focus on extending CloudSim capabilities to address the requirements of use cases. This includes further model development, as well as integration with data collection framework and optimisation framework. Also, the preliminary simulation performance test (described in Chapter 5.1) reveals high CloudSim memory dependency and underutilisation of full CPU resources due to sequential event queue processing. To improve the performance of RECAP simulation platform, strategies of system model granularity reduction and event queue parallelisation of will be investigated. To parallelise the simulation process we are planning to analyse the benefits of integration with the parallelised CloudSim modification known as NetworkCloudSim (Garg & Buyya, 2011), as well as exploring the use of the MPI based simulator from CloudLightning H2020 project (Cloud Lightning Consortium, 2017) which has undergone its first open source release in December 2017.

The initial version of the RECAP Simulator is available online[3] and a public news announcement of the release being available has been issued.[4]

The source code will be released publicly during the project using the future RECAP source code management system.

7 References

Alves, D. C., Batista, B. G., & Filho, D. M. (2016). CM Cloud Simulator: A Cost Model Simulator Module for Cloudsim. *Services (SERVICES), 2016 IEEE World Congress.* San Francisco, CA, USA: IEEE.

Buyya, R., Ranjan, R., & Calheiros, R. N. (2009). Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. *High Performance Computing & Simulation, 2009. HPCS '09.* Leipzig, Germany: IEEE.

Byrne, J., Svorobej, S., Giannoutakis, K., Tzovaras, D., Byrne, P. J., Östberg, P.-O., . . . Lynn, T. (2017). A Review of Cloud Computing Simulation Platforms and Related Environments. *CLOSER*, (pp. 679-691).

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience, 41*, 23-50.

Cloud Lightning Consortium. (2017). D7.1.1 Large scale modelling and simulation framework.

Garg, S. K., & Buyya, R. (2011). Networkcloudsim: Modelling parallel applications in cloud simulations. *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, (pp. 105-113).

Iorga, M., Feldman, L., Barton, R., Martin, M. J., Goren, N., & Mahmoudi, C. (2017). The NIST Definition of Fog Computing. *NIST Special Publication, 800*, 13.

Lynn, T., Gourinovitch, A., Byrne, J., Byrne, P. J., Svorobej, S., Giannoutakis, K., . . . Morrison, J. (2017). A Preliminary Systematic Review of Computer Science Literature on Cloud Computing Research using Open Source Simulation Platforms.

Medina, A., Lakhina, A., & Matta, I. (2001). BRITE: an approach to universal topology generation. *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001.* Cincinnati, OH, USA, USA: IEEE.

Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. *NIST Special Publication, 800*, 7.

Mihaela-Catalina Nita, F. P. (2014). FIM-SIM: Fault injection module for CloudSim based on statistical distributions. *Journal of Telecommunications and Information Technology* , 14-23.

[3] https://recap-project.eu/about/public-deliverables/
[4] https://recap-project.eu/news/initial-simulation-platform/

Östberg, P.-O., Byrne, J., Casari, P., Eardley, P., Anta, A. F., Forsman, J., . . . others. (2017). Reliable capacity provisioning for distributed cloud/edge/fog computing applications. *Networks and Communications (EuCNC), 2017 European Conference on*, (pp. 1-6).

RECAP Consortium. (2017). D3.1 Initial Requirements.

RECAP Consortium. (2017). D8.1 Initial infrastructure modelling and resource mapping.

Rodrigo N. Calheiros, R. R. (2009). *CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services.* Australia: arXiv.

Sá, T. T., Calheiros, R. N., & Gomes, D. G. (2014). CloudReports: An extensible simulation tool for energy-aware cloud computing environments. In *Cloud Computing* (pp. 127-142). Springer.

Shi, W., Cao, J., & Zhang, Q. (2016). Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal (Volume: 3, Issue: 5, Oct. 2016)*, 637 - 646.