CHAPTER 5

Simulating Across the Cloud-to-Edge Continuum

Minas Spanopoulos-Karalexidis, Christos K. Filelis Papadopoulos, Konstantinos M. Giannoutakis, George A. Gravvanis, Dimitrios Tzovaras, Malika Bendechache, Sergej Svorobej, Patricia Takako Endo, and Theo Lynn

Abstract As growth and adoption of the Internet of Things continue to accelerate, cloud infrastructure and communication service providers (CSPs) need to assure the efficient performance of their services while meeting the Quality of Service (QoS) requirements of their customers and

M. Spanopoulos-Karalexidis • C. K. Filelis Papadopoulos • K. M. Giannoutakis • D. Tzovaras
Information Technologies Institute, Centre for Research and Technology Hellas, Thermi, Greece
e-mail: mspanopoulos@iti.gr; kgiannou@iti.gr; dimitrios.tzovaras@iti.gr

G. A. Gravvanis
Democritus University of Thrace, Xanthi, Greece
e-mail: ggravvan@ee.duth.gr

T. Lynn et al. (eds.), *Managing Distributed Cloud Applications and Infrastructure*, Palgrave Studies in Digital Business & Enabling Technologies, https://doi.org/10.1007/978-3-030-39863-7_5

their end users, while maintaining or ideally reducing costs. To do this, testing and service quality assurance are essential. Notwithstanding this, the size and complexity of modern infrastructures make real-time testing and experimentation difficult, time-consuming, and costly. The RECAP Simulation Framework offers cloud and communication service providers an alternative solution while retaining accuracy and verisimilitude. It comprises two simulation approaches, Discrete Event Simulation (DES) and Discrete Time Simulation (DTS). It provides information about optimal virtual cache placements, resource handling and remediation of the system, optimal request servicing, and finally, optimal distribution of requests and resource adjustment, with the goal to increase performance and concurrently decrease power consumption of the system.

Keywords Simulation • Discrete event simulation • Discrete time simulation • Resource allocation • Capacity planning • CloudSim • CloudLightning Simulator • Distributed clouds

5.1 Introduction

In this chapter, an overview of the RECAP Simulation Framework which comprises two different simulation approaches is outlined and discussed—Discrete Event Simulation (DES) and Discrete Time Simulation (DTS). The RECAP Simulation Framework offers two approaches to recognise the characteristics, requirements, and constraints of cloud and

M. Bendechache • S. Svorobej (✉)
Irish Institute of Digital Business, Dublin City University, Dublin, Ireland
e-mail: Malika.Bendechache@dcu.ie; sergej.svorobej@dcu.ie

P. T. Endo
Irish Institute of Digital Business, Dublin City University, Dublin, Ireland

Universidade de Pernambuco, Recife, Brazil
e-mail: patricia.endo@upe.br

T. Lynn
Irish Institute of Digital Business, DCU Business School, Dublin, Ireland
e-mail: theo.lynn@dcu.ie

communication service providers. As such, the RECAP Simulation Framework offers a solution for (1) SMEs and large hyperscale cloud and network operators, and (2) providers requiring rapid less-detailed simulation results and those requiring a more-detailed simulation.

DES focuses on aggregating each incoming request in the form of events, regarding their entry timestamp, and usually in a pipelined manner. These events are stored in an initialised list of tasks, retained in memory and augmented with each incoming task. In order to accommodate all this information, the required resources and especially memory requirements are significant and large. Thus, DES is suitable for simulating smaller and intensively detailed scenarios, in order to maintain accuracy at high levels. DTS on the other hand provides the potential to simulate larger scenarios with its ability to scale up significantly. This is feasible due to the fact that DTS does not need precomputation and storage of future events; it uses a time-advancing loop, where the requests are entering the system in respective time steps during the simulation. This results in a significant reduction in memory requirements, providing significant improvements in the ability to scale up the simulation. DTS does not offer the level of detail of DES, but it can be a useful and accurate tool for simulating real large-scale scenarios, while maintaining resource consumption on reasonable levels.

In this chapter, a high-level overview of the RECAP Simulation Framework is presented and discussed. This is followed by a brief overview of the RECAP DES framework, followed by a short case study illustrating its applicability for cloud infrastructure and network management. Then, the RECAP DTS framework is presented with a short case study illustrating its applicability for simulating virtual content distribution networks.

5.2 High-Level Conceptual Overview of the RECAP Simulation Framework

The RECAP Simulation Framework facilitates reproducible and controllable experiments to support the identification of targets for the deployment of software components and optimising deployment choices before actual deployment in a real cloud environment. It was designed specifically to simulate distributed cloud application behaviours and to emulate data centre and network systems across the cloud-to-thing continuum.

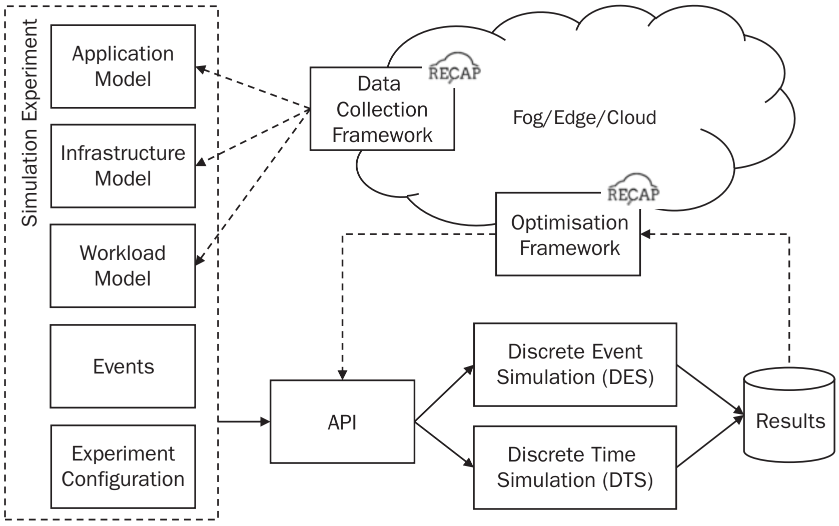

Fig. 5.1 High-level conceptual overview of the RECAP simulation framework

Figure 5.1 presents a high-level conceptual overview of the RECAP simulation framework comprising the following components: Simulation Experiment Models, Application Programming Interface (API), Simulation Manager, and Model Mappers for DES and DTS simulators.

The system monitoring data obtained from the RECAP Data Analytics Framework (presented in Chap. 2) are used to compile the Simulation Experiment Models:

- Application Model: represents the application components and their connections and behaviour, i.e. application load propagation and operational model;
- Infrastructure Model: describes the physical infrastructure (network topology and (physical and virtual) machines' configurations) where the application will be hosted;
- Workload Model: describes how the workload generated by the users is distributed and processed by the application components; and,
- Experiment Configuration: where the simulation user configures the simulation parameters, such as simulation time, simulation time step, log files, and input files.

The RECAP Optimisation Framework makes use of the RECAP Simulation Framework to evaluate different application deployments and infrastructure management alternatives in terms of cost, energy, resource allocation, and utilisation, before actuating on real application deployments. This integration is done through an API that receives the models, the experiment configuration, and the set of simulation scenarios, and sends them to a web-based REST API. Depending on the type of API call, the experiment is forwarded to the RECAP DES or DTS simulator. Once simulation is completed the results can be accessed from the chosen storage method, e.g. local CSV files or a database.

5.3 Discrete Event Simulation

5.3.1 Overview

Discrete Event Simulation (DES) is a system modelling concept wherein the operation of a system is modelled as a chronological sequence of events (Law et al. 2000). DES-based decision support processes can be divided into three main phases: modelling, simulation, and finally, results analysis. During the modelling phase, a simulated system is defined by grouping interacting entities that serve a particular purpose together into 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 in sequence. An event represents actions happening in the system during operation time. Depending on the event type, the system reaction is simulated, and associated metrics captured. These metrics are collected at the end of the simulation for results analysis. Therefore, system behaviour can be examined under different conditions. Using DES is beneficial in a complex real non-deterministic small-to-medium-sized system environment (SME) where the system definition using mathematical equations may no longer be a feasible option (Idziorek 2010).

5.3.2 The RECAP DES Framework

The RECAP DES Framework captures system configurations by using Version 3 of the Google Protocol Buffers technology.[1] This implementation approach was chosen to ensure model schema would

[1] https://developers.google.com/protocol-buffers/

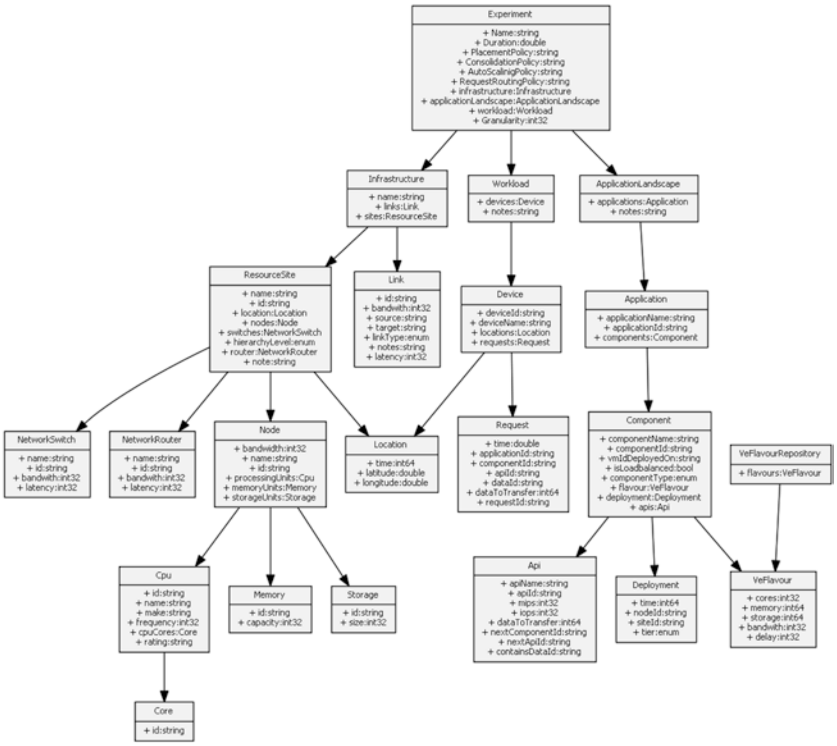

Fig. 5.2 DES simulation model data format (inputs)

remain programming language-neutral allowing serialised models to be used across multiple language platforms such as Java, C++, C#, or Python. In addition, Protocol Buffers are simple and faster to use and are smaller in size compared to XML or JSON notations. Speed and file size of the model are important when dealing with large-scale systems by managing the memory footprint of simulation framework and sending data over a RESTful web API client.

As shown in Fig. 5.2, the RECAP DES simulator root element is the *Experiment* class which contains nested system models, the name identifier of the simulation experiment, and its parameters used by the simulation engine, i.e. *Duration, Granularity, PlacementPolicy, ConsolidationPolicy,* *AutoScalingPolicy* and *RequestRoutingPolicy*. The *Duration* parameter defines the length of the simulation experiment in simulated time while the *Granularity* is a multiplier for the number of requests represented by a single simulation event. Placement, consolidation autoscaling, and request routing policies are optional attributes which specify the name of any resource management policies which can be integrated within the simulator. In addition, the *Experiment* class contains nested *Infrastructure*, *Workload*, and *ApplicationLandscape* models, which describe edge to cloud system composition and behaviour.

The *Infrastructure* model captures the hardware characteristics of a distributed network and computes hardware locations. Each *ResourceSite* component in the model represents a virtualised cloud/edge/fog datacentre location which is geographically distributed with *Location* class containing latitude and longitude spatial information. Nested *NetworkSwitch*, *NetworkRouter* and *Node* model components capture network bandwidth, latency, and compute resource (CPU, memory, storage) capacity at each location.

The *Workload* model contains mappings between devices and requests devices made to the system. The *Device* system component has *Name* and *ID* attributes as well as a time-dependent *Location* array and an array of *Requests*. Each *Request* component describes a request of the device (user) made to the system at a specific geographical location. The *Request* attributes capture the time of the request, amount of data to transfer, type of data, and application model API where the request is destined for.

The *ApplicationLandscape* component contains information on the applications running in the virtualised infrastructure. Each application can be composed of multiple interconnected components, and each application component can have multiple functions expressed through an API definition; hence in the model, we have *Application*, *Component*, and *API* classes describing the relationships. The model assumes a one-to-one relationship between the application component and a Virtual Entity (VM or Container) it is deployed to. Therefore, the *Component* class also contains the *Deployment* class describing which hardware node it is deployed to and a *VeFlavour* class specifying what resources it requires.

The simulation results, called outputs, are also arranged in a structured form using Protocol Buffers. The proposed format structure is captured within a class diagram shown in Fig. 5.3.

The *ExperimentResult* root class splits into two arrays of simulated system behaviour metrics: *ResourceSiteMetrics* and *ApplicationMetrics*. As the

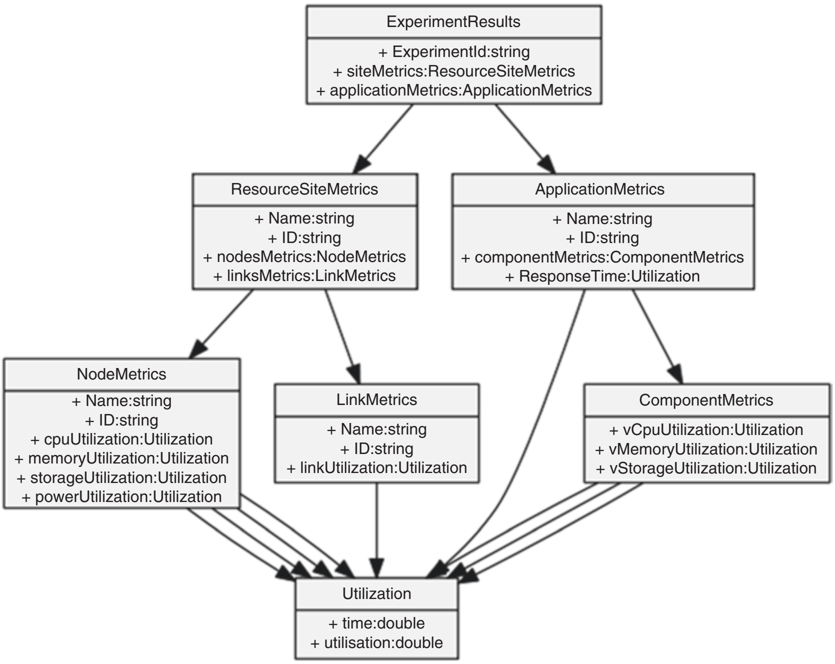
ExperimentResults
+ ExperimentId:string
+ siteMetrics:ResourceSiteMetrics
+ applicationMetrics:ApplicationMetrics
ResourceSiteMetrics
+ Name:string
+ ID:string
+ nodesMetrics:NodeMetrics
+ linksMetrics:LinkMetrics
ApplicationMetrics
+ Name:string
+ ID:string
+ componentMetrics:ComponentMetrics
+ ResponseTime:Utilization
NodeMetrics
+ Name:string
+ ID:string
+ cpuUtilization:Utilization
+ memoryUtilization:Utilization
+ storageUtilization:Utilization
+ powerUtilization:Utilization
LinkMetrics
+ Name:string
+ ID:string
+ linkUtilization:Utilization
ComponentMetrics
+ vCpuUtilization:Utilization
+ vMemoryUtilization:Utilization
+ vStorageUtilization:Utilization
Utilization
+ time:double
+ utilisation:double

Fig. 5.3 DES simulation results format (outputs)

name suggests the *ResourceSiteMetrics* class contains information on hardware utilisation and *ApplicationMetrics* contains information on application performance metrics. The subclass *NodeMetrics* stores per-node metrics of CPU, memory, storage, and power utilisation. The subclass *LinkMetrics* stores each link utilisation bandwidth. Similarly, the subclass *ComponentMetrics* captures utilisation of resources per individual virtual entity besides including a response time metrics for end-to-end application performance in the upper *ApplicationMetrics* class. All of the measurements are captured at regular time intervals, hence attributes in the *Utilisation* class list time and the actual utilisation value. The *Utilisation* class is built as an abstract and can be extended to fit different types of measurements.

Once the models are created based on the desired system parameters, they can be then loaded into the RECAP DES Simulator. The RECAP

DES Simulator is based on CloudSim[2] with a custom DES implementation in the back-end. To load, read, and query the simulation input models and output results shown in Figs. 5.2 and 5.3, Google Protocol Buffers library provides auxiliary methods ensuring ease of use.[3]

5.3.3 *Cloud Infrastructure and Network Management: A RECAP DES Framework Case Study*

To illustrate how the RECAP simulation and modelling approach can be used by communication service providers, we present the application of the RECAP Simulation Framework for mobile technology service management within fog/cloud computing infrastructure. This case study is based on automated services and infrastructure deployment (using virtual network functions (VNF)), automated orchestration, and optimisation services to reach the desired QoS for the different network services. We model distributed infrastructure and a VNF service application chain using the RECAP DES Framework.

5.3.3.1 Infrastructure Model

The infrastructure simulation model was designed and implemented to capture available physical characteristics of real edge infrastructure and used input from the infrastructure models described in Chap. 4. It consists of several sites that are interconnected by links between each other. Each *Site* entity in the model represents a location that is hosting network and/or computing equipment, such as switches, routers, and computing nodes. *NetworkSwitch* and *NetworkRouter* capture attributes of bandwidth and latency while *Nodes* in addition to bandwidth also capture properties of CPU, Memory, and Storage. For the simulation experiments, physical infrastructure for 45 distributed sites was modelled; each site contains a router for handling inbound and outbound internet traffic and two switches handling control plane and user plane traffic separately. This meant that any traffic that is received or transmitted from the site is traversing through the router and internal traffic between physical hosts and is flowing through routers only. The user plane switch was assigned 40 Gbps bandwidth and control plane switch 1 Gbps where routers were

[2] http://www.cloudbus.org/cloudsim/

[3] The methods are well documented in tutorials widely available for a range of programming languages: https://developers.google.com/protocol-buffers/docs/tutorials

assigned 100 Gbps. Bandwidth assumptions were made based on the data gathered from testbed experiments and correspond to the volume of traffic observed. In addition, links between sites were assigned additional latency delays proportionate to the distance between locations; hence, requests sent between sites take more time to arrive.

5.3.3.2 Application and Workload Propagation Model

Application behaviour for the simulation is realised by implementing a modelling concept that captures data flow through multiple interconnected, distributed, components. Each component is represented as a virtual entity (VM or container) that is assigned to a physical machine in a site and has access to the portion of resources.

Application behaviour logic is realised through multiple interconnected API elements that each component has. The API represents a model object that holds information on resource demand and connection to the next component in line, thus forming a logical path between different application components. The current use case is based around NFV paradigm, and the used VNF chain is of a virtualised LTE stack which consists of user data plane and control data plane virtual components:

- eNodeB user plane denoted as ***CU-C***
- Mobility Management Entity control plane denoted as ***MME-C***
- eNodeB control plane denoted as ***CU-U***
- Serving Gateway-User plane denoted as ***SGW-U***
- Packet Data Network Gateway-User plane denoted as ***PGW-U***

For example, Fig. 5.4 graphically describes one of the possible application topologies where application components like CU-C, CU-U, SGW-U, and PGW-U are located on one site and component MME-C is located on another site. In this example, both user plane download (green) and upload (red) requests are executed on one site, but the control plane (blue) requests require to travel to another site to be processed resulting in longer processing delays. When a request arrives at a component, based on the API parameters number of resources are requested from the hardware to process this request. Once the appropriate amount of resources is available, the request is sent further in the system according to the API connection path. During the simulation experiments, 1482 different application configurations (placements) were generated and combined with the infrastructure model and corresponding workload models.

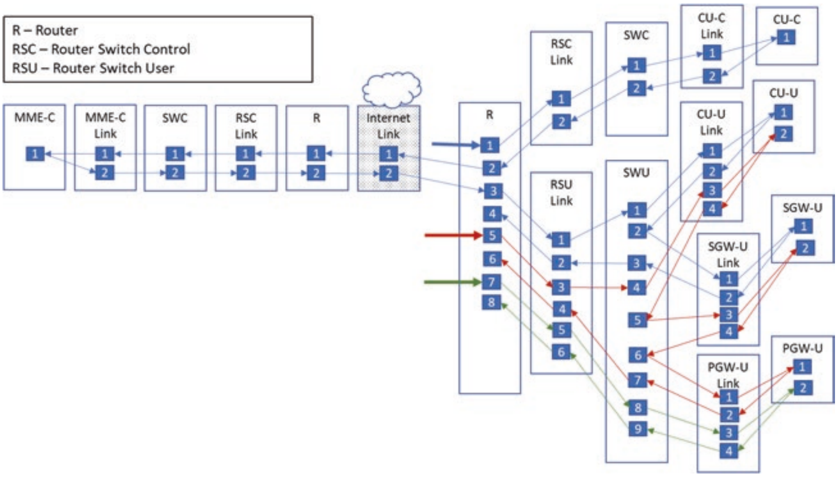

Fig. 5.4 Application simulation model example

The workload simulation model was implemented to capture the number of end-user devices that use the system for the duration of the simulation experiment. The model contains an array of devices each containing multiple requests. Furthermore, each request contains information on the arrival time, the size of the request, and application information for its destination. The number of users varies depending on the VNF placement on the infrastructure sites. More densely populated areas have more users, and this aspect is reflected in the workload models; hence, each placement experiment has a bespoke number of users. User-request parameters were based on the data gathered from testbed experiments, and an average of the quantity and size of the request was done, along with defining them into three categories *User Download*, *User Upload*, and *Control*. As shown in Table 5.1, on average data download request in user plane is 13,927 bits and user makes 2808 requests per hour. For data upload requests user sends around 224 requests per hour each of 8572 bits in size and finally control plane administrative requests were taken as a fracture of upload traffic and amount to 6 requests per hour each 219 bits in size.

DES simulations were executed in concurrent batches of 5 parallel runs on a dedicated VM in a testbed. The VM configuration was set to 8 CPU cores, 64 GB of RAM, and 500 GB attached volume storage. Each

Table 5.1 Average values of user requests

Request type	*Requests per user per hour*	*Request size (bits)*
User download	2808	13,927
User upload	224	8572
Control	6	219

simulation experiment was set for 3600-second duration of simulated time and on average took around 30 seconds of wall clock time to complete.

The simulation results were analysed using utility functions with the total resource utilisation and the cost of the allocated machine serving as the provider utility with equal weighting, along with the network bandwidth consumed and total latency serving as the customer utility with equal weighting. The mathematical formulation of these utility functions is available in Chap. 4. Our goal was to minimise resource utilisation utility, latency utility, and cost utility while maximising network bandwidth utility. The total utility for the placement was then defined as an equally weighted sum of normalised provider and customer utility.

5.3.3.3 RECAP DES Results

Results showed that by fixing the provider utility or customer utility, there is further scope to maximise the corresponding utility by changing the placement distribution of the VNFs. This is highly beneficial for stakeholders when making business decisions regarding the available infrastructure. These decisions map back to the values considered within the definition of both utility functions such as response time, bandwidth, latency, and utilisation costs.

5.4 Discrete Time Simulation

5.4.1 Overview

Discrete Time Simulation (DTS) is a simulation technique based on a time-advancing loop of predefined starting and ending time. The defined time step is a portion of time values (usually seconds) that the user has the ability to set before the execution. During each time step, potential new requests/events enter the system from the defined entry points. The major advantage of DTS is that no precomputation and storage of future events are needed, thus resulting in a significant reduction in memory consumption requirements. This also suggests the possibility to dynamically allocate simulated resources based on current computational load.

5.4.2 The RECAP DTS Framework

The RECAP DTS simulation framework is based on the CloudLightning Simulation Platform, which is designed to simulate hyperscale heterogeneous cloud infrastructures (Filelis-Papadopoulos et al. 2018a, b). It was built using the C++ programming language utilising OpenMP to exploit parallelism and acceleration in computations where applicable. The RECAP Simulation Framework focuses on optimally placing VMs as caches or containers in a network while taking into account efficient resource utilisation, reduction of energy consumption and end-user latency, and load balancing for minimisation of network congestion.

The CloudLightning Simulation Platform was developed to simulate hyperscale environments and efficiently manage heterogeneous resources based on Self-Organisation Self-Management (SOSM) dynamic resource allocation policies. The simulated cloud architecture is based on the Warehouse Scale Computer (WSC) architecture (Barroso et al. 2013). It manages to maintain a simplistic approach by utilising models that do not demand extremely high computational effort and, at the same time, maintain accuracy at adequate levels. The utilisation of a time advancing loop, rather than a discrete sequence of events, enables the potential to use these dynamic resource allocation techniques while also providing high scalability due to the lack of restrictions in memory requirements.

A brief summary of the basic characteristics of the CloudLightning Simulation architecture is as follows. The gateway lies at the topmost level on the master node and the cells, which are connected directly to the gateway, hosted on separate distributed computing nodes at a lower level. Each cell is responsible for the underlying components, such as cell's broker, network, telemetry, and finally, hardware resources. Key responsibilities of the gateway are (1) communication with the available cells, in the essence of data transport, fragmentation, and communication of the task queues between the cells with the appropriate load balancing on each time step; and (2) receiving and maintaining metrics and cells' status, amongst others. From the cell's perspective, the key responsibilities are (1) the aforementioned communication with the gateway, including simulation parameters and initialisation of the underlying components, and additionally, sending status and metrics' information to the gateway; and (2) task queue receipt on each time step, finding the optimal component with the required available resources, utilising the SOSM engine, and finally, executing the tasks.

Considering the above, many CloudLightning Simulator components were adopted for the RECAP Simulation Framework including power consumption modelling, resource utilisation (vCPU, memory, storage), and bandwidth utilisation. Some of the most important differences between the two frameworks are: (1) the focus of the RECAP Simulator which is on the optimal cache placement in the network, and (2) the difference in task servicing, and specifically in task deployment to the available nodes. The CloudLightning Simulator utilises a Suitability Index formula and is based on the required weights communicated by the gateway to the underlying components. The most appropriate node is assigned with the incoming task, by adopting a first-fit approach. The RECAP Simulator, on the other hand, utilises caches with the corresponding content placed in the network. In order to assign a task to a respective node, it performs a search for the optimal available node, which offers, in excess, the required resources while adequately handling network congestion. Experimental results of the CloudLightning Simulator demonstrated that it can accurately handle simulations of hyperscale scenarios with relatively low computational resources. This is particularly suitable for large distributed networks that many Tier 1 network operators manage.

5.4.3 Network Function Virtualisation—Virtual Content Distribution Networks: A RECAP DTS Case Study

Traditional Content Distribution Network (CDN) providers occasionally install their hardware, such as customised hardware caches, in third-party facilities or within the network of an Internet Service Provider. BT has such a scenario, which the RECAP DTS simulator utilises as a case study. BT's main activities focus on the provision of fixed-line services, broadband, mobile and TV products and services, and networked IT services as well. BT hosts customised hardware caches from the biggest CDN operators in their network. Considering the fact that it would be extremely hard, for sensitive reasons, for content providers to install their hardware in many locations across the UK in the edge nodes of BT's network (also known as Tier 1 MSANs (Multi-Service Access Nodes)), there lies the need to provide an alternate solution. In order to ensure the required QoS for their virtual network functionalities, the introduction of a Virtual CDN (vCDN) provides a beneficial approach, which aims to replace the presence of multiple physical caches in the network, with standard servers and storage providing multiple virtual applications per CDN operator. BT accomplishes that, by installing the appropriate compute infrastructure at its edge nodes (MSANs) and thus offering a CDN-as-a-Service (CDNaaS). In this way, operational costs are significantly declined and additionally, the content is stored in virtual caches closer to end user, thus minimising end-user latency and maximising user experience.

5.4.3.1 DTS Architecture and Component Modelling

The topological architecture is divided in four tiers, in a hierarchical order namely MSAN, Metro, Outer-Core, and Inner-Core, in a total number of 1132 nodes; more information on infrastructure architecture is provided in the next subsection. Note that in order to maintain efficient simulation accuracy, a specific time interval is selected, at which all the components update their status. This provided the opportunity to reduce computational cost, but it is essential to mention that the choice of the interval value is critical. A small interval can lead to huge computational effort and reduce performance, while a large interval can lead to major accuracy leaks, considering that whole requests could be missed during the status update process. Considering all these, the RECAP DTS framework provides the essential scalability for the current use case. The ambition is to improve the efficiency of vCDNs systems by replacing multiple customised physical caches running multiple virtual applications per CDN operator.

Figure 5.5 depicts the DTS architecture optimised to simulate a vCDN network. The ***Graph Component*** is responsible for the input topology of the simulation, which is fed to the component as an input file in Matrix Market storage format. The structure is stored as a Directed Acyclic Graph (DAG) in the component, in the form of a sparse matrix, with the number of rows being the total number of sites and each row, the ID of a site. More specifically, it is stored both as a Compressed Sparse Row (CSR) and Compressed Sparse Column (CSC) format which results in a faster traverse of the available connections. These connections are indicated by off-diagonal values and point to links with lower level sites, while the diagonal values denote the level/tier of the respective site.

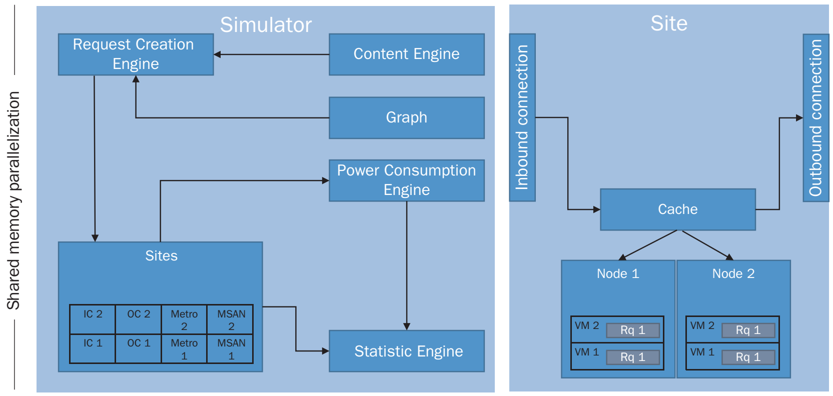

Fig. 5.5 DTS architecture

All the available sites are retained in a *vector of sites*, where a site is an object of the Site class which contains a number of attributes. Each site has a unique ***ID*** value and a ***type*** value, indicating the tier of the respective site. Furthermore, a two-dimensional vector retains the available connections ***input*** and ***output*** to the immediate upper and lower level respectively. Note that in case of the first level there are no input connections, and similarly, there are no output connections in case of the last level. Another vector retains the ***output bandwidth*** of the output connections as double precision values (Gbps). All the attached nodes, of predefined type, to the site are also retained in a vector and contain resource information such as *CPU*, *Memory*, and *Storage*. These nodes are mostly utilised by the power consumption component. In addition to nodes, there is a list of the hosted VMs deployed to the nodes of each site and provide same information with the addition of *Network*. Additionally, a map, which contains the ***cached content*** in the VMs, is used in order to simplify and speed up the search of specific content type or available VMs. All the ***cache hits and misses*** are also stored into a vector; these refer to all active VMs hosted by a single site. Lastly, a site can ***forward requests*** to sites at a higher tier, due to lack of VMs or insufficient resources in them. These forwarded requests are also retained in a list and have an impact only on the network bandwidth of the site.

The ***Content Component*** retains all the potential information referring to the type of content a VM in a site can serve. Specifically, this information includes minimum and maximum duration of each type of content and the requirements that need to be met by the VM in order to serve it. Also, the probability of a cache hit for each type of content and maximum number of requests that can be served are included. The requirements of each type of cached content request is provided by the ratio of the VM requirements of this specific request to the aforementioned maximum number of requests the VM can serve.

The requests are generated by the ***Request Creation Engine***, which is responsible for the insertion of a group of requests to the system in each time step. This component is based on a uniform distribution generator, which produces requests of each type of content and duration between a given interval denoting the minimum and maximum requests permitted in a time step. Each request contains the following information: *duration* of the request, *type* of content, and the *site* from which it enters the system. For each of the inserted requests, a path (list of sites) is formed showing the flow the content will follow in order to reach the user. During path creation, each site traversed and is appended to the end of the list; this continues until a cache hit occurs or otherwise. If the last element is not a cache hit, the request is rejected. When a cache hit occurs, the content flows downwards from that site to all sites of the path of a lower tier until it reaches the user.

During each time step, the duration of all requests is reduced until it reaches zero, at the point they are considered served and can be discarded from the system, thus freeing up resources. This procedure takes place in each site as well as in the site's nodes and VMs that update their status. This is where OpenMP provides acceleration of the computations on shared memory systems. Each site is independent; thus, they can be assigned to the available threads and their status updated without any interference. During each time step, each site checks its current status, duration of requests, any new additions, or any finished, and respectively adjusts available resources. This procedure has no data traces, as sites are independent. Thus, this computational-intensive process, considering the number of sites and the huge number of time steps can be performed in parallel, saves significant amount of time and increases performance. Apart from status update, at each time step, another component is also used, the ***Power Consumption Component***. This calculates the power/energy consumption of the site's nodes depending on their type (Makaratzis et al. 2018).

Power consumption along with other metrics is stored in the last component, the ***Statistics Engine*** which is deployed at specific time intervals. It contains all metrics of the vCDN network and each site generally. Metrics include the aforementioned power consumption, cache hits and misses, and other stats per level such as cumulative accepted and rejected requests per level, average vCPU, Memory, Storage, and Network utilisation per level. The Statistics Engine outputs these metrics to files at each specific time interval. Note again that accurate selection of this interval is critical otherwise it can lead to either huge writing effort (in the case of a small interval) or under-sampling (in the case of a large interval).

5.4.3.2 Infrastructure Model

The considered vCDN system is hierarchical and has sites located at four different levels: (1) inner core, (2) outer-core, (3) metro, and (4) Multi-Service Access Node (MSAN), as illustrated in Fig. 5.6.

The physical network topology is composed of 1132 sites. Each site can host physical machines (nodes) that, in turn, can host vCDNs containing content requested by customers. Moreover, each site has predefined upload and download bandwidth as well as inbound and outbound connections. The general structure of a site is given in Fig. 5.7.

Each node inside a site can host multiple VMs and each VM services specific content. However, multiple VMs can service the same content if it

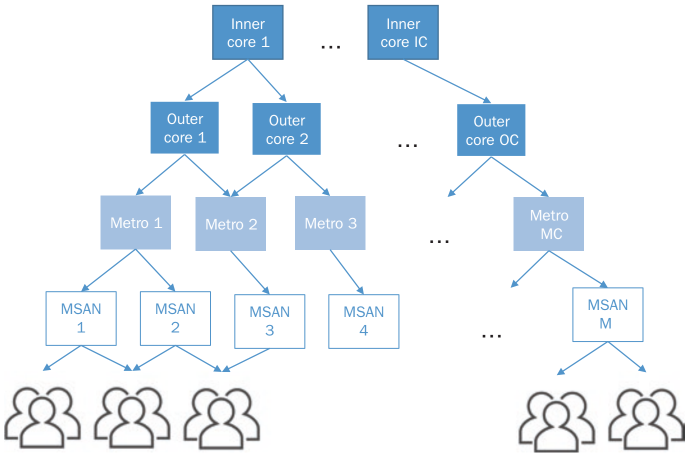

Fig. 5.6 BT hierarchical level of sites

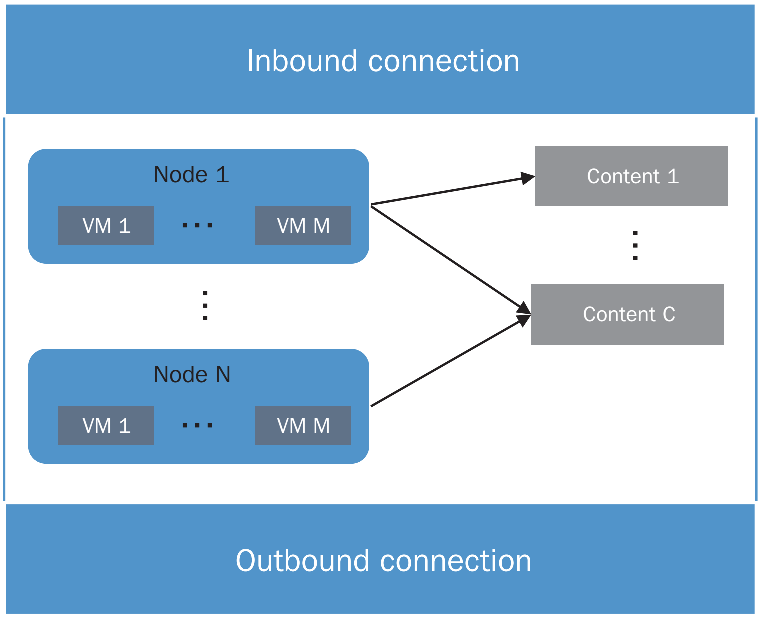

Fig. 5.7 A site architecture of DTS

is requested by a large number of users, since each VM has predefined capacity to service customer.

5.4.3.3 Application and Workload Propagation Model

Each type of content (content with different sizes) characterises a VM. For example, content of type *j* is serviced by a specific VM with predefined characteristics described above. Thus, each type of content has its unique accompanying requirements. Moreover, each user of a specific content requires predefined bandwidth and occupies the system for a variable amount of time lying between predefined intervals. For each type of content, the requirements in terms of VMs as well as per user bandwidth are defined. The same content can be hosted in several VMs on the same site, since the number of users requiring a specific type of content serviced by a VM is limited. The characteristics of content include required vCPUs per VM, required memory per VM, required storage per VM, maximum number of customers per VM, network bandwidth required per VM at full capacity.

Regarding the creation of VMs, the DTS should offer two options:

1. **Static**: In this case, no new VM can be created during a simulation. This requirement comes from Infrastructure Optimiser that will use the RECAP Simulation Framework to decide placements; or
2. **Dynamic**: In this case, every time a cache miss occurs at MSAN site, the requested content is copied to that given MSAN site. Moreover, when there is no user requesting a given content, this content (VM) is deleted from the node. If the node has no more available resources, then the request is rejected.

The Request Creation Engine (RCE) creates a series of requests based on random number generators following a preselected distribution such as Uniform, Normal, or Weibull. Each request performed by a user is considered to have predefined requirements with respect to content. Thus, all customers of a certain content type require the same amount of resources. However, customers requiring different content require a different amount of resources.

5.4.3.4 RECAP DTS Results

The results of this simulation are presented in detail in C. K. Filelis-Papadopoulos et al. (2019). In summary, we find that parallel performance (status update) is significantly increasing proportionally to the number of requests. In addition, resource consumption seems to reach stability, for all levels, by the time initial requests have finished execution. The lowest level contributes the most to resource and node underutilisation through request forwarding to upper layers as a result of probabilistic caching and single VM hosting. This leads to reduced active server utilisation and increased power consumption concurrently due to node underutilisation. Nevertheless, energy consumption is improved with the reduction of VMs on the lowest level; thus, these sites act as forwarders to the immediate upper layer. This impacts efficient resource utilisation in the upper layers, while they service more requests forwarded from the bottom tier.

Other experiments focused on different probabilities for a cache hit, such as 0.4 and 0.8. The former leads to requests servicing from the topmost layer due to the fact that content requested is not potentially cached in the lower layers and thus a cache miss occurs and the requests are forwarded. The latter increases the probability for a cache hit to occur and thus requests are serviced mostly from lower layers. More specifically, the intermediate level serves a significantly increased amount of tasks. Note that, with 0.8 probability the energy consumption is considerably decreased when compared to the other two cases. Nevertheless, high probability denotes that the content is cached in a great portion of the distributed caches in the network, as the probability value acts as a mechanism to transfer workload between the corresponding nodes and tiers of the network. Thus, potential deployment of virtual caches of specific content in great numbers could result in higher costs and storage requirements. On the other hand, lower probability denotes a significant reduction in virtual cache numbers, especially on lower levels. As discussed earlier, this results in higher service rates from the top layers and furthermore in potential network congestion due to increased data traffic in the links of the network, request rejection, and increased end-user latency in request servicing from nodes significantly further from the end users.

Finally, we performed experiments with an increased number of levels. The scalability performance results suggest that the simulator scales linearly with the number of input requests, considering the major increase (mostly two times) in memory requirements. The results illustrate that the framework is capable of executing large-scale simulations in a feasible time period even with significant memory requirements (as number of threads increases, the need of memory for local data storage also increases) and at the same time maintaining required high levels of accuracy. Thus, the RECAP DTS framework can be a useful tool for content providers to validate their overall performance.

5.5 Conclusion

In this chapter, the RECAP Simulator Framework, comprising two simulation approaches—DES and DTS, was presented. The design and implementation details of the RECAP simulation framework were given in both simulation approaches, coupled with case studies to illustrate their applicability in two different cloud and communication service provider use cases. The main advantage of this framework is the fact that depending on the target use case requirements, an appropriate simulation approach can be selected based on a time-advancing loop or a discrete sequence of events. Thus, by providing this flexibility, focus can be given on the level of accuracy of the results (DES) or the scalability and dynamicity (DTS) of the simulation platform.

From the experimentation performed, the RECAP simulation platform was capable of efficiently simulating both discrete event and discrete time use cases thus providing a useful tool for non-data scientists to forecast the placement of servers and resources by executing configurable prediction.

References

Barroso, L.A., J. Clidaras, and U. Hoelzle. 2013. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool. https://doi.org/10.2200/S00516ED2V01Y201306CAC024.

Filelis-Papadopoulos, C.K., K.M. Giannoutakis, G.A. Gravvanis, and D. Tzovaras. 2018a. Large-scale Simulation of a Self-organizing Self-management Cloud Computing Framework. *The Journal of Supercomputing* 74 (2): 530–550. https://doi.org/10.1007/s11227-017-2143-2.

Filelis-Papadopoulos, C.K., K.M. Giannoutakis, G.A. Gravvanis, C.S. Kouzinopoulos, A.T. Makaratzis, and D. Tzovaras. 2018b. Simulating Heterogeneous Clouds at Scale. In *Heterogeneity, High Performance Computing, Self-organization and the Cloud*, 119–150. Cham: Palgrave Macmillan.

Filelis-Papadopoulos, Christos K., Konstantinos M. Giannoutakis, George A. Gravvanis, Patricia Takako Endo, Dimitrios Tzovaras, Sergej Svorobej, and Theo Lynn. 2019. Simulating Large vCDN Networks: A Parallel Approach. *Simulation Modelling Practice and Theory* 92: 100–114. https://doi.org/10.1016/j.simpat.2019.01.001.

Idziorek, Joseph. 2010. *Discrete Event Simulation Model for Analysis of Horizontal Scaling in the Cloud Computing Model*. Proceedings of the 2010 Winter Simulation Conference, 3003–3014. IEEE.

Law, Averill M., W. David Kelton, and W. David Kelton. 2000. *Simulation Modeling and Analysis*. New York: McGraw-Hill.

Makaratzis, Antonios T., Konstantinos M. Giannoutakis, and Dimitrios Tzovaras. 2018. Energy Modeling in Cloud Simulation Frameworks. *Future Generation Computer Systems* 79 (2): 715–725. https://doi.org/10.1016/j.future.2017.06.016.