Investigating IoT Application Behaviour in Simulated Fog Environments

Andras Markus and Attila Kertesz
Department of Software Engineering, University of Szeged, Szeged, Hungary
{markusa,keratt}@inf.u-szeged.hu

Abstract. In the past decade novel paradigms appeared in distributed systems, such as Cloud Computing, Fog Computing and the Internet of Things (IoT). Sensors and devices of IoT applications need big data to be stored, processed and analysed, and cloud systems offer suitable and scalable solutions for them. Recently fog nodes are utilized to provide data management functionalities closer to users with enhanced privacy and quality, giving birth to the creation of IoT-Fog-Cloud systems. Such infrastructures are so complex that they need simulators for planning, designing and analysis. Though cloud simulation already has a large number of literature, the simulation of fog systems is still evolving. In this paper we plan to take a step forward in this direction by investigating current fog simulation approaches and compare two of them providing the broadest fog modeling features. We also perform evaluations of executing IoT applications in hybrid, Fog-Cloud architectures to show possible advantages of different setups matching different IoT behaviour.

Keywords: Fog computing · Internet of Things · Cloud computing · Simulation

1 Introduction

Parallel and distributed computing went through a rapid evolution in the past decade giving birth to cloud and fog technologies enabling virtualized service provisions. The appearance of small computational devices connected to the Internet has led to the Internet of Things (IoT) paradigm, which resulted in a vast amount of data generations requiring the assistance of cloud services for storage, processing and analysis. Cloud systems become good candidates to serve IoT applications, and their marriage created so-called smart systems [1]. One of their latest improvements addresses data locality meaning that data management operations are better placed close to their origins to reduce service latency. This idea created Fog Computing [2], which implied the appearance of IoT-Fog-Cloud systems with the highest complexity.

This research was supported by the Hungarian Government and the European Regional Development Fund under the grant number GINOP-2.3.2-15-2016-00037 (Internet of Living Things), and by the Hungarian Scientific Research Fund under the grant number OTKA FK 131793, and by the Hungarian Government under the grant number EFOP-3.6.1-16-2016-00008. This paper is a revised and extended version of the conference paper [14].

© Springer Nature Switzerland AG 2021
D. Ferguson et al. (Eds.): CLOSER 2020, CCIS 1399, pp. 258–276, 2021.
https://doi.org/10.1007/978-3-030-72369-9_11
These IoT-Fog-Cloud systems require significant investments in terms of design, development and operation, therefore the use of simulators for their investigation is inevitable. There are a large number of simulators addressing the analysis of parts of these systems, and we can find survey papers of cloud, IoT and fog simulators summarizing their basic capabilities and comparing them according to certain metrics, e.g. by [3]. These surveys highlight that fog modelling in simulators still needs further research, despite the promising approaches already capable of examining IoT-Fog-Cloud systems to some extent. In this paper we compare two of such simulators, namely the iFogSim and DISSECT-CF-Fog, which are able to model fogs, and found to be reliable and widespread enough by former surveys.

The main contributions of our paper are the comparison of the fog modelling capabilities offered by iFogSim and DISSECT-CF-Fog, and the evaluation of their features with a meteorological IoT application with three rounds of evaluations representing an increasing level of complexity. This paper is a revised and extended version of a conference paper [14]. We added new measurements and performance comparisons of the simulators.

The rest of the paper is structured as follows: Sect. 2 introduces the related literature in fog simulation. Section 3 presents a comparison of the fog modelling capabilities of the selected two simulators. Section 4 introduces the configurations for the experiments, while Sects. 5, 7 and 8 presents the evaluation rounds of an IoT application with different scenarios representing different complexity. Finally, Sect. 9 concludes our work.

2 Related Approaches in Fog-Cloud Simulation

We can find several survey papers in the field of Cloud Computing and Fog Computing of tools supporting modelling and simulation. Concerning the properties and modelling of Fog Computing, Puliafito et al. [3] presented a survey highlighting and categorizing the properties of Fog Computing, and investigated the benefits of applying fogs to support the needs of IoT applications. They introduced six IoT application groups exploiting fog capabilities, and gathered fog hardware and software platforms supporting the needs of these IoT applications. Markus et al. [11] focused on available cloud, IoT and fog simulators, and compared them according to several metrics such as software metrics and general characteristics. Concerning fog simulation, they introduced and classified 18 simulators.

Table 1. Comparison of fog simulators as partly shown in [14].

| Simulator       | Based on       | Published Year | Type             | Hits | Citations | Language |
|-----------------|----------------|----------------|------------------|------|-----------|----------|
| DISSECT-CF-Fog  | DISSECT-CF     | 2019           | Event-driven     | 82   | 79        | Java     |
| EdgeCloudSim    | CloudSim       | 2019           | Event-driven     | 183  | 127       | Java     |
| YAFS            | —              | 2018           | Event-driven     | 1    | 17        | Python   |
| FogNetSim++     | OMNET++        | 2018           | Network          | 3    | 37        | C++      |
| iFogSim         | CloudSim       | 2017           | Event-driven     | 851  | 664       | Java     |
| DockerSim       | iCanCloud      | 2017           | Network          | 10   | 6         | C++      |
| EmuFog          | —              | 2017           | Emulator         | 59   | 49        | Java     |
We selected seven recent fog simulators, and briefly compared them in Table 1. We noted their base simulator, publication date and type for their categorization. The network type simulators usually focus on low-level network interaction between entities such as routers, switches and nodes, but less suitable for the higher level of abstraction (e.g. virtual machines), whilst event-driven type simulators are more general and usually lack implemented the network operations or only support minimal network traffic simulation, but they are easier to be used for accurate representation of higher level system components. We also summarized the number of literature search results (i.e. hits) performed in Google Scholar [16], and we summed the number of citations of the top five relevant hits. We also listed the applied programming language of the simulators.

DISSECT-CF-Fog is based on DISSECT-CF, and a direct extension of the DISSECT-CF-IoT simulator [13], also developed by the authors. The base simulator is able to model cloud environments and supports energy measurements of physical resources. The extended version supports the modelling of IoT systems and its communications. The whole software is fully configurable, and follows a hierarchical structure.

EdgeCloudSim [5] is a CloudSim extension with the main capabilities of network modelling, including extensions for WLAN, WAN and device mobility. The developers of this tool aimed to respond to the disadvantage of the simple network model of iFogSim by introducing network load management and content mobility to this simulator.

YA F S [6] simulator is proposed to investigate application deployment (i.e. module placement) on fog topology. It also supports the modelling of user mobility and dynamic failures of nodes and the implementation of different scheduling and routing algorithm of the IoT tasks.

The FogNetSim++ [7] is built on the OMNeT++ discrete event simulator, which focuses on network simulation. This extension provides configuration options for fog network management including node scheduling and selection. It is also able to model different communication protocols, such as MQTT or CoAP, and different mobility models.

One of the most applied and referred fog simulators is iFogSim [8], which is based on CloudSim. iFogSim can be used to simulate cloud and fog systems using the sensing, processing and actuating model. It is able to model cloud and fog devices with certain resource parameters. Sensors and actuators can also be managed represented by a Tuple. There are dedicated modules for processing and data-flows.

DockerSim [9] aims to support the analysis of container-based SaaS systems in simulated environments. It is based on the iCanCloud network simulator, this extension can model container behaviour, network, protocol and OS process scheduling behaviour.

EmuFog [10] emulator is dedicated to design fog infrastructures and investigate real application and workloads by emulation. EmuFog also handles its own network topology model and components (e.g. routers) and similarly to the DockerSim, it uses Docker containers for the application components deployed on fog nodes during the execution.

Though all of these simulators would be interesting to be further analysed, after performing a quick pre-evaluation we found that iFogSim and DISSECT-CF-Fog are the most mature and documented solutions, and we also took into account a literature
search result and number of citations for our decision. We also considered numerous iFogSim extensions, which have appeared in the last few years, and the support for novel functions or properties of Fog Computing (as proposed by a recent survey in [11]). Unfortunately, only a few of those extensions were published with available source code, thus our goal was to make a comparison with the original version of iFogSim in a comprehensive way.

3 Fog Modelling in Two Simulators

The CloudSim-based extensions (e.g. iFogSim or EdgeCloudSim) are often used for investigating Cloud and Fog Computing approaches, and in general they are the most referred works in the literature. On the other hand, the DISSECT-CF simulator is proven to be much faster, scalable and reliable than CloudSim (see [4]). This work showed that the simulation time of DISSECT-CF is 2800 times faster than the CloudSim simulator for similar purely cloud use cases, therefore we have chosen to analyse their latest extensions to compare fog modelling. Next, we introduce the main properties of these simulators, and compare their fog modelling capabilities.

iFogSim is a Java-based simulator, its main physical components are the following: (i) fog devices (including cloud resources, fog resources, smart devices) with possibility to configure CPU, RAM, MIPS, uplink- and downlink bandwidth, busy and idle power values; (ii) actuators with geographic location and reference to the gateway connection; (iii) sensors, which generate data in the form of a Tuple representing information. The main logical components aim to model a distributed application: the AppModule, which is a processing element of iFogSim, and the AppEdge that realises the logical data-flow between the VMs. The main management components are: the Module Mapping that searches for a fog device to serve a VM – if no such device is found, the request is sent to an upper tier object; and the Controller is used to execute the application on a fog device. For simulating fog systems, first we have to define the physical components, then the logical components, finally the controller entity. Although numerous articles and online source codes are available for the usage of this simulator, there is a lack of source code comments for many methods, classes and variables. As a result, application modelling with this tool requires a relatively long learning curve, and its operations take valuable time to understand.

DISSECT-CF-Fog is an discrete event simulator for modelling Cloud, IoT and Fog environments, written in Java programming language. The main advantage of this tool is the detailed configuration possibilities across its low-level components: timer modules manage the simulation time and the events. The network layer can be used for simulating bandwidth and latency, and it models data transfers as well. The physical components are responsible for the creation of a physical infrastructure of any graph hierarchy with storage support for resource and file modelling. The sensor and smart device layer is responsible for modelling data generation with a certain frequency, measurement delays, geographical position and network connections, and sensor configurations. The application layer handles the physical topology, the task mapping for VMs, and the data-flow between the physical components. Finally, the support layer is capable of applying pricing models of real cloud providers to calculate resource usage costs.
for the experiments. These parameters could be easily edited through XML configuration files, thus large scale simulation experiments can be executed even without Java programming knowledge (for the predefined scenarios).

Table 2. Comparison of DISSECT-CF-Fog and iFogSim as partly shown in [14].

| Property                        | DISSECT-CF-Fog                  | iFogSim                   |
|--------------------------------|--------------------------------|---------------------------|
| Unit of the simulation time     | Tick                           | Millisecond               |
| Unit of the processing          | CPU core processing power      | MIPS                      |
| Physical component              | ComputingAppliance             | FogNode                   |
| IoT model                       | Device and Sensor              | Sensor                    |
| Logical component               | Application                    | Application with AppModule and AppEdge |
| Task                            | ComputeTask on VM              | Tuple                     |
| Architecture                    | Graph                          | Tree                      |
| Communication direction         | Horizontal and vertical        | Vertical                  |
| Data-flow                       | Implicit in physical connection | Separately in the AppEdge |
| Sensor                          | Processing depends on the size of the data generated | Predefined MIPS value |
| Cloud Pricing                   | Price per tick for each VM     | Static cost for RAM, storage, bandwidth and CPU |
| IoT Pricing                     | Using real IoT providers’ schemas | N.A.                     |

iFogSim and DISSECT-CF-Fog are quite evolved and complex simulators, and follow different logic to model Fog Computing, as the previous paragraphs highlighted. This means that though they have similar components, we cannot match them easily. Based on [8], iFogSim was created to model resource management techniques in Fog environments, for what the DISSECT-CF-Fog can also be applicable [12]. To facilitate their comparison, we gathered and compared their properties and components closest to each other as it can be seen in Table 2. Its first column names a generic simulation property or entity, the second column shows how they are represented in DISSECT-CF-Fog, and the third summarizes their representation in iFogSim. As we can see, the biggest difference between them is the chosen unit for simulation time measurement. iFogSim measures time passing in the simulated environment in milliseconds, while DISSECT-CF-Fog has a specific naming for the smallest unit for simulation time called tick, which is related to the simulation events. The researcher using the simulator can set up the parameters and properties of a concrete simulation to associate a certain time interval (e.g. millisecond) for a tick. The measurement of processing power in the simulators can also be done with different approaches. iFogSim associates MIPS for every node, which represents the computational power and does not take into account the number of CPU cores. The number of CPU cores affects only the creation of virtual machines. In DISSECT-CF-Fog both physical machines (PM) and virtual machines (VM) have to be configured with CPU core processing values, which define how many instructions should be processed during one tick.

A physical component is represented by one dedicated class (see 3rd row of Table 2) in both simulators. To represent IoT components, iFogSim uses the Sensor class, while DISSECT-CF-Fog differentiate general IoT devices with computing and storage capacities and smaller sensors managed Device and Sensor classes. The logical components to define concrete applications are implemented with three classes defining processing elements and logical data-flow in iFogSim (Application, AppEdge, AppModule),
which are not straightforward to configure. Besides, the ModuleMapping class is an important component, which is responsible for the mapping of the logical and physical entities based on a given strategy (e.g. cloud-aware, edge-aware). On the other hand, in DISSECT-CF-Fog the physical topology already defines data routes, so researchers can focus on setting up the required processing units (of the components placed in the topology). The representation of computational tasks is also different. In DISSECT-CF-Fog, researchers should define a ComputeTask with a certain number of instructions, also stating the number of instructions to be executed within a tick. In iFogSim, researchers should define a so-called Tuple for each task and state the number of MIPS required for its execution. In DISSECT-CF-Fog tasks can be dynamically created to process a certain amount of sensor-generated data, therefore the number of instructions will be proportional (to the available data) in the created tasks. In iFogSim a static MIPS value should be defined in the Tuple, hence it cannot respond to the actually generated data of a scenario. Concerning the communication among components, iFogSim orders components in a hierarchical way and supports only vertical communication among elements of its layers (by default), while DISSECT-CF-Fog supports communication to any direction among any components in the topology. To support cost calculations and pricing, in iFogSim static cost can be defined for CPU, bandwidth, storage and memory usage. DISSECT-CF-Fog has a more mature cost model, and it supports XML-based configuration for cloud and IoT side costs based on real provider pricing schemes.

To achieve fair comparison of the two simulators, we apply the following restrictions. We limit the configuration of DISSECT-CF-Fog by allowing only single core CPUs for the simulated resources. In case of DISSECT-CF-Fog, the speed of the task execution depends on the number of CPU cores and processing power of those, whilst in the iFogSim only the MIPS value of the task defines the time of task processing, as we mentioned before. The common parameters that can be set up in both simulators with similar values are the followings: simulation time, data generation frequency, processing power and configuration of the physical resources, count of instructions for the tasks, and finally the physical topology. Nevertheless, we cannot avoid introducing some different setups. In iFogSim, the devices have direct connections to the physical resources, while in DISSECT-CF-Fog, connection properties also include actual coordinates and distances to the corresponding physical resources.

We also have to deal with the issue that iFogSim does not take into account the size of the generated data in task creation, because the Sensors in iFogSim always create Tuples with the same MIPS value, hence the file size does not have an influence on that value. As a result, dynamically received sensor data on a fog node cannot be modelled, only static, predefined tasks have to be used. To allow fair comparison, we configured the scenarios in DISSECT-CF-Fog to always generate task with the same size. Concerning task forwarding, in iFogSim a fog device uses a method to forward a received (or generated) task to a higher-level device, if it cannot handle (i.e. process) it. In case of DISSECT-CF-Fog, every application module has a threshold value to handle task overloading, which defines the number of allowed waiting tasks. If this number exceeds the threshold (so more tasks arrive than it could be processed), the unhandled tasks will be forwarded to other available nodes (according to some selection algorithm). To match the default behavior of iFogSim, the topology defined in DISSECT-CF-Fog allowed
only vertical forwarding among the available fog nodes (i.e. tasks are forwarded to upper nodes only).

4 IoT Application Scenarios and Architecture Configuration for the Evaluation

After applying the configuration restrictions for the two simulators discussed in the previous section, we define the IoT application scenarios to be used for comparison. Since meteorological applications are commonly used in IoT [13], we define our scenarios in this field. In our notion sensors are attached to IoT devices, which are weather stations that monitor weather conditions, and send the sensed data to fog or cloud resources for weather forecasting and analysis.

To perform the comparison, we defined four layers for the topology: (i) a cloud layer, (ii) an upper fog device layer with stronger resources, (iii) a lower fog device layer with weaker resources, and (iv) an IoT (smart) device layer. For the concrete resource parameters we defined one scenario with three different test cases:

- In the first test case we set up 20 IoT devices to generate data to be processed;
- in the second test case we initiated 40 IoT devices;
- while in the third test case we initiated 60 IoT devices for data generation (where each device had a single sensor).
- Concerning data processing we used the following resource parameters for the test cases: one cloud with 45 CPU cores and 45 GB RAM, 4 (stronger) fog nodes with 3 CPU cores and 3 GB RAM each, 20 (weaker) fog nodes with 1 CPU core and 1 GB RAM.

We did not use preset workloads for the experiments, only the started sensors generated data independently, thus in both simulators we executed so-called bag-of-tasks applications in fogs and clouds. In this work we refrain from distinguishing containers and traditional virtual machines, hence both considered simulators model virtual machines to serve application execution.

5 Results of the First Evaluation Round

To be as close to iFogSim as possible, we only used one type of Virtual Machine in DISSECT-CF-Fog, having 1 CPU core and 1 GB RAM. In case of iFogSim, the power of virtual machines was 1000 MIPS. The tasks to be executed in VMs were statically set to 2500 MIPS in both simulators. The simulation time was set to 10 000 s, and sensor readings were done every 5.1 s (i.e. the data generation frequency of the sensors). Each sensor generated 500 bytes of data during one iteration. The latency and bandwidth values were set equally in both simulators.

All the experiments were run on a PC having Intel Core i5-7300HQ 2.5 GHz, 8 GB RAM and running a 64-bit Windows 10 operating system. The results of executing the first round test cases with both simulators are shown in Table 3. We executed the same test cases five times with both simulators and counted their medium values to be stored
in the table. To compare the use of the simulators, we only took into account the default outputs of the simulators and their execution time (e.g. cost calculations were neglected, hence they follow different logic in the simulators, and also do not really relevant for the performance comparisons).

According to these measurements, we can observe that the time needed for executing the simulation of the first test case was about ten times more with iFogSim, than with DISSECT-CF-Fog. In the second test case we doubled the number of IoT devices, and the runtime values increased with about 25% in case of DISSECT-CF-Fog and about 71% in case of iFogSim.

Comparing their runtime, DISSECT-CF-Fog is better suited for high-scale simulations, while iFogSim simulations become intolerably time consuming by modelling higher than a certain number of entities. In the third test case we could not even wait the measurements to finish (cancelled them after 1.5 h).

The application delay is the time within the simulation needed to process all remaining data in the system, after we stopped data generation by the IoT devices. The results in Table 3 show that this delay was longer in case of iFogSim, though the generated data sizes were equal for the same test cases in both simulators (hence the output results concerning the processed data were also equal). This is due to the different methods of task creation, scheduling and processing in the simulators (we could not eliminate all differences with the restrictions).

Finally, we used a simple source code metric to compare the implemented scenarios in the simulators. The so-called Lines of Code (LOC) is a common metric for analysing software quality. It is interesting to see that the same scenario could have been written three times shorter in case of DISSECT-CF-Fog, than in iFogSim. Of course, we tried to implement the code in both simulators with the least number of methods and constructs (in Java language). We also have to state that some configuration parameters had to be set at different parts of the software (this adds some lines in case of iFogSim, and around 20 lines of XML generation and configuration in case of DISSECT-CF-Fog). The considered iFogSim scenario is available online [17], while the DISSECT-CF-Fog scenarios are available here [18].

With this evaluation round we managed to model an IoT-Fog-Cloud environment with both simulators, and investigated a meteorological IoT application execution with different sensor and fog and cloud resource numbers. While DISSECT-CF-Fog dealt these simulations with ease, iFogSim struggled to simulate more than 65 entities of this complex system. Nevertheless, it is obvious that there are only a small number of real-world IoT applications that require only hundreds of sensors and fog or cloud resources; we need to be able to examine systems and applications composed of hundred thousands

### Table 3. Comparison of the two simulators.

| Property               | DISSECT-CF-Fog | iFogSim     |
|------------------------|----------------|-------------|
| Test case              | I.  | II. | III. | I.   | II. | III. | |
| Runtime (ms)           | 248.75 | 312.5 | 392.58 | 2260.33 | 3873.66 | 5400000* |
| Application delay (min)| 3.41  | 4.33 | 4.33  | 14.89 | 17.52 | N.A. |
| Generated data (byte)  | 19600000 | 39200000 | 58800000 | 19600000 | 39200000 | N.A. |
| Lines of Code          | 50 lines + 6 XML files for detailed configuration | 159 lines + 11 inline constants |
Table 4. Software metrics of the investigated simulators.

| Metric            | DISSECT-CF-Fog | iFogSim |
|-------------------|----------------|---------|
| Lines of Code     | 8.7k           | 28k     |
| Duplication (%)   | 1.0            | 24.4    |
| Code Smells       | 512            | 1.8k    |
| Vulnerabilities   | 12             | 3       |
| Bugs              | 37             | 139     |
| Language          | Java, XML      | Java, XML |
| Files             | 108            | 291     |
| Classes           | 160            | 306     |
| Comments (%)      | 35.8           | 25.2    |
| Cognitive Complexity | 1.695        | 4.122   |

of these components. We continue our investigations in this direction, and we further raise the scale and analyse the behavior of DISSECT-CF-Fog, after we analyse thorough the system utilisation of the simulators in the next section.

6 Further Analysis of the Investigated Simulators

Since the result of the first scenario showed strong performance difference between the investigated simulators, we decided to perform further examinations on the source code of these simulators. To this end we used a static code analyzer tool and a performance profiler software, that helped us to point out implementation differences of the simulators. First, we used the SonarCloud tool [19] that provides static code metrics, security and code quality information, which are strongly related to the intuitive understanding of the code.

The results shown in Table 4 are based on the following information: Lines of Code refers to the number of lines in the source code of the software (empty lines are omitted), Duplication shows the percentage of the same lines in the code, Code Smells are defined as a maintainability-related component, which increases the time of changing the code (e.g. empty statements or readability of “switch-if” statements). Vulnerabilities are strongly connected to security issues (e.g. visibility of class members). Bugs refer to something wrong in the code, which require immediate fixes or corrections (e.g. variable casting or avoiding division by zero). Language shows the used programming and markup language, the Files and Classes represents the number of these elements in the code, respectively. Comments help the understanding of the source code and the software itself, thus it presents the ratio of the total lines (including empty lines) and lines containing comments. Finally, the Cognitive Complexity is an often used metric, which defines the understandability of methods. The higher the Cognitive Complexity is, the longer time and higher efforts are needed to understand the code and work with it.

Concerning the results, iFogSim contains about three-times more Lines of Code and about twice more Files and Classes, than DISSECT-CF-Fog. Except for the Vulnerabilities, the number of the Code Smells and Bugs are much higher in case of iFogSim, similarly to the Duplication ratio. These values show that there is a potentially higher chance to product malfunction during an iFogSim simulation. The Cognitive Complexity is about 2.5 smaller in case of DISSECT-CF-Fog, and it reaches better results in
Next, we applied JProfiler [20], which is able to analyse Java-based applications considering threads, classes, instances, usage of the garbage collector, besides memory and CPU usage. We repeated the test cases defined for both simulators in the previous section focusing on these metrics and characteristics.

Concerning the results, first we looked at the memory allocation tree, and the aggregation level that was set to the methods. In case of iFogSim, the `startSimulation` method is responsible for starting the simulation and the `SimEntity` class represents entities, which handle events. In case of DISSECT-CF-Fog, the `SimulateUntilLastEvent` starts the simulation and `Timed` manages the inner clock of the simulator. Accordingly, a simulation in the iFogSim utilises 7.2 times higher memory, than DISSECT-CF-Fog, as shown in Fig. 1.

![Memory allocation call tree of iFogSim simulator](image1)

![Memory allocation call tree of DISSECT-CF-Fog simulator](image2)

**Fig. 1.** Memory allocation call tree of the simulators.
We also investigated the telemetries of the scenarios, as shown in Fig. 2. Each row presents the three test cases of first scenario in the corresponding simulator. The comparison is based on the following characteristics: Memory reflects the heap memory, the used size labeled by blue, whilst the free size of memory is labeled by green. Recorded Objects present the instantiated objects, blue refers to the number of arrays, green refers to the non-arrays objects. The Recorded Throughput shows the freed object per seconds using green colour, and blue represents created objects per second. The GC Activity presents the activity ratio of the garbage collector of the JVM, Thread presents the number of threads with runnable state, whilst the Classes shows the number of used classes during the evaluation. Finally, CPU Load reflects to the process load (by green) and the system load (by blue).

Interpreting the results, DISSECT-CF-Fog utilises less memory, and in all test cases the heap size stays less than 200 MBs, whilst in case of iFogSim the heap size of the second test case almost reaches the 400 MBs. The Recorded Object value is almost four-times higher during the evaluation with iFogSim, however DISSECT-CF-Fog uses almost 3000 Java classes for the evaluation (external libraries are considered by the JProfiler, as well). The iFogSim tool uses the CPU more intensively, than DISSECT-CF-Fog, the CPU Load almost reaches 90% in the first two test cases during the iFogSim simulations. The GC Activity and the Recorded Throughput metrics point out a possible malfunction in the third test case of iFogSim, because after about 5 s, these values are correlated showing no relevant operation occurrence.

The reason of this behaviour is the process starvation caused by the Java Finalizer thread. Similar issue is mentioned in [15], however this problem can strongly related to the negative code quality as well.
For the second round of evaluation we extend our investigation for larger IoT systems and applications, thus we increase the number of fog nodes to hundreds and smart devices to hundreds of thousands. We found DISSECT-CF-Fog more reliable for managing fog environments in our previous evaluation round, we continue its investigation with six additional test cases, in which we compare a cloud-centred solution to a fog-centred architecture, where additional fog nodes appear beside cloud resources.

As we mentioned before, DISSECT-CF-Fog uses an XML document structure to configure system parameters. To define the additional scenarios, we need to know this structure. Figure 3 presents an example of such description, which contains only one physical fog infrastructure (called \textit{appliance}), but its tag can be used multiple times in the document. The \textit{name} tag is the unique identifier of a fog device, and the \textit{xcoord}, \textit{ycoord} describes the exact location of this physical resource. In this case this XML describes a child fog node, since the \textit{parentApp} refers to its parent node, which is a cloud apparently. The \textit{file} tag contains the absolute path of another XML file, which present the configuration of physical machines this fog node should have. The \textit{application} tag is also repeatable, it tells what kind of application this physical resource has.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example_xml.png}
\caption{Sample XML description for the application model in DISSECT-CF-Fog as shown in [14].}
\end{figure}

7 The Second Evaluation Round
Table 5. Maximum number of created entities during the simulations in iFogSim and DISSECT-CF-Fog.

| Scenario | DISSECT-CF-Fog | iFogSim |
|----------|----------------|---------|
|          | Cloud/Fog nodes | IoT devices | Sensors | Cloud/Fog nodes | IoT devices | Sensors |
| I/a      | 25             | 20       | 20      | 25             | 20       | 20      |
| I/b      | 25             | 40       | 40      | 25             | 40       | 40      |
| I/c      | 25             | 60       | 60      | 25             | 60       | 60      |
| II/a     | 3              | 100 000  | 500 000 | N.A.           |          |         |
| II/b     | 98             | 100 000  | 500 000 |                |          |         |
| II/c     | 113            | 100 000  | 500 000 |                |          |         |
| II/d     | 153            | 100 000  | 500 000 |                |          |         |
| II/e     | 208            | 100 000  | 500 000 |                |          |         |
| II/f     | 152            | 100 000  | 500 000 |                |          |         |

(should execute). The tasksize attribute tells us how much data (in bytes) should be gathered to create a task (250 kB in this example). appName is the unique identifier of this application module. The application has a task creation frequency (called freq), which defines periodical intervals for task generation and data forwarding (in this case its value is 300000 ms, i.e. five minutes). The instance tag refers to a VM type this application should use. Finally, one can define possibly multiple neighbouring devices (by stating a formerly defined unique identifier of an infrastructure in the device tag), to which data or tasks may be forwarded. Possible advantages using XML files are to create simulation that researchers do not have to understand the tasks of low-level simulator components, XML schemas can secure more readable format to configure the system than a Java code.

In the current evaluation phase we introduced different VM types of flavors, to show some of the additional capabilities DISSECT-CF-Fog has. We used three real VM pricing and configuration types based on Amazon Web Services’ offerings [21]: (i) a1.large VM (1 CPU cores, 2 GBs RAM with $0.051 hourly cost), (ii) a1.xlarge VM (2 CPU cores, 4 GBs RAM with $0.102 hourly cost) and the last one is (iii) a1.2xlarge VM (8 CPU cores, 16 GBs RAM with $0.204 hourly cost).

In this phase we enabled the dynamic task creation method that takes into account the size of the generated data. Our default configuration (for clouds) required every task to contain maximum 2 500 000 bytes of data (to be processed).

We defined IoT-Fog-Cloud systems using the same four layers as we used before: (i) a cloud layer, (ii) a stronger fog layer (called Fog Type 1 – T1), (iii) a weaker fog layer (called Fog Type 2 – T2), and (iv) an IoT device layer. In each layer we could define different number of cloud or fog infrastructures with different resources. We used the following configuration values for the computational infrastructures:

- a cloud contains 200 CPU cores and 400 GBs RAM, and all of its VMs are of type a1.2xlarge.
- a T1 fog contains 48 CPU cores and 112 GBs RAM, offering a1.xlarge VM type. It also redefines the default task size (to be executed in this node) to 1 250 000 bytes.
- a T2 fog contains 12 CPU cores and 24 GBs RAM, offering a1.large VM type. The task size in this infrastructure is set to 625 000 bytes.
We also changed the configuration of IoT devices (weather stations in the analogy) in this phase. Instead of containing a single sensor, we defined five sensors to be attached to an IoT device (so a weather station has five sensors to monitor temperature, humidity, pressure, rain, wind speed), all of them generating 100 bytes of data every minute (which is the data generation frequency).

In DISSECT-CF-Fog one can set the maximum number of tasks to be handled by a computational node. In this evaluation phase we set it to three, so if more data arrived to a node than what three tasks could process, the remaining data is forwarded (neighbouring) fog or cloud node to be processed. In case there is no available VM to execute a newly created task on a node, the VM managed tries to deploy a new one.

We used six different topologies for this second scenario: (a) three clouds, (b) three clouds with 15 Fog T1 and 80 Fog T2, (c) three clouds with 30 Fog T1 and 80 Fog T2, (d) three clouds with 30 Fog T1 and 120 Fog T2, (e) three clouds with 45 Fog T1 and 160 Fog T2 and the last (f) two clouds with 50 Fog T1 and 100 Fog T2.

To reach hundreds of thousands of simulated components we created eight test cases for each topology defined earlier. We investigated how the system behaves under serving an increased number of IoT devices. We defined 5 000 smart devices (weather stations) at the beginning, and we scaled them up to reach 10 000, 20 000, 30 000, 40 000, 50 000, 75 000, and finally in the last test case the total number of IoT devices were 100 000 (each of them run with five sensors, thus our simulator managed 500 000 entities). Table 5 summarizes the number of simulated components (entities) in each of the performed experiments.

8 Result of the Third Evaluation Round

Finally, for the third round of evaluations, we scaled up the simulation time to 24 h of weather forecasting, while we run the simulated data generation and processing for around 2.7 h in the former two rounds.

We present the evaluation results by comparing each scenario with the following metrics: the number of IoT devices managed in the simulations (we recall that each device had five sensors that generated data), the number of VMs needed to process the generated data, the total costs of operating the IoT devices and utilizing the VMs both in fogs and clouds, the application delay (or timeout) values that denoted the time passed after stopping the sensors (i.e. its data generation) till the end of the simulation, and finally the runtime (execution time) of the actual simulation. Detailed evaluation results are depicted in Figs. 4, 5, 6, 7 and 8.

Figure 4 shows an important characteristic of our simulated system. The configuration of larger task sizes in clouds led to the creation of a relatively small number of strong VMs (75 in the largest test case) to process these tasks. In case of fogs we had a much higher number of nodes executing a large number of weaker VMs (834 in the largest test case) to process the larger number of tasks (created to process less data than in clouds). We can also notice that after the fifth case (managing 40 000 IoT devices) the number of cloud VMs does not grow any more: we reached the maximum capacity of the available clouds (all physical resources are fully utilized). This means that in the purely cloud cases the infrastructure was heavily overloaded during managing
more than 40,000 devices (weather stations). It can be also observed that the fog-aware topology \( f \), which includes less cloud nodes but more Fog T1 nodes utilizes less VMs in average (433.81) than the other fog topologies (516.57).

This issue was also approved by the results as it can be seen in Fig. 5, where we can observe the costs to be paid for hiring the management infrastructure. The purely clouds scenarios were the cheapest, where a small number of expensive VMs were
Correlation of application delay and the number of managed IoT devices in DISSECT-CF-Fog simulations. Figure 6 reveals additional interesting behavior. The f topology is slightly faster than the purely cloud topology in the first test case, but it then starts to increase, similarly to the other fog topologies. This figure also shows that in the purely cloud scenario the overloading started even after utilizing more than 20 000 IoT devices. So managing around 25 000 IoT devices (i.e. 125 000 sensors) we can see a trend break point: it is faster and cheaper to manage less number of devices with only clouds, while for a higher number of devices utilizing fogs can help to reduce the application delay (with higher costs). One can also observe that the f topology tries to keep the application delay as below as possible, the average delay is 347.0 min, while the average of the rest of the fog scenarios is 593.7 min.

For the test case having the highest scale, 6 591 min were needed for the application to terminate (after data generation of the sensors was stopped) in the purely cloud scenario, while utilizing the largest fog infrastructure needed only 2 574 min. By correlating this with their costs, we can conclude that we have to pay about 46% more for about 156% faster data processing.
Our last figure reveals how the simulator coped with the test cases of the scenarios. Figure 7 presents the elapsed real time (wall-clock time, or simply runtime) taken to execute the simulations of the test cases. It is also interesting to see that simulating a higher number of fog and cloud nodes and VMs took less time than a smaller number of cloud nodes and VMs. We can observe that runtime is in correlation with the application delay, so this is one of the reasons for this issue. The other explanation is that in the fog cases the higher number of smaller tasks (and their data) were better treated (processed) by the higher number of fog VMs, while in the purely cloud cases many bigger tasks (with larger amount of data) had to be waiting in queues for the overloaded VMs.
To summarize our evaluation, we can conclude that DISSECT-CF-Fog has a more detailed and fine-grained fog model than iFogSim as the comparative analysis shows us. It can also scale up to simulate hundreds of thousands of IoT-Fog-Cloud system components simultaneously with acceptable runtime. Our experiments also revealed that utilizing fogs beside clouds can be beneficial in terms of reducing the application execution time (and delay in our notion), though we had to pay more for them. Nevertheless, different pricing schemes for fogs (other than clouds) may also result in cost savings (e.g. own or a neighbor’s fog device may be free to use in smart home applications).

9 Conclusions

The recent technological advances have transformed distributed systems and enabled the creation of complex environments called IoT-Fog-Cloud systems. The design, development and operation of these systems need simulation tools to save costs and time, therefore specialized simulators need to be created to provide means to investigate these processes.

In this paper we investigated the available solutions in this field, and compared two fog modelling approaches, namely iFogSim and DISSECT-CF-Fog, with detailed evaluations in three rounds of experiments representing an increasing level of complexity. The evaluations also showed how to create and execute simulated IoT scenarios using fog and cloud resources with these tools.

Our results highlight that DISSECT-CF-Fog can provide easier configurations and faster and more reliable simulations for higher scales, but the benefits of utilizing fog or cloud resources are highly dependent on the actual behaviour of the considered IoT application.

Our future work will investigate a more detailed representation and use of mobility features of IoT and fog devices.

References

1. Botta, A., de Donato, W., Persico, V., Pesca pie, A.: Integration of cloud computing and Internet of Things: a survey. Future Gener. Comput. Syst. 56 (2016). https://doi.org/10.1016/j.future.2015.09.02
2. Dastjerdi, A.V., Buyya, R.: Fog computing: helping the Internet of Things realize its potential. Computer 49 (2016). https://doi.org/10.1109/MC.2016.245
3. Puliafito, C., Mingozzi, E., Longo, F., Puliafito, A., Rana, O.: Fog computing for the Internet of Things: a survey. ACM Trans. Internet Technol. 19 (2019). https://doi.org/10.1145/3301443
4. Mann, Z.A.: Cloud simulators in the implementation and evaluation of virtual machine placement algorithms. Softw. Pract. Exp. 48 (2018). https://doi.org/10.1002/spe.2579
5. Sonmez, C., Ozgovde, A., Ersoy, C.: EdgeCloudSim: an environment for performance evaluation of Edge Computing systems. In: Second International Conference on Fog and Mobile Edge Computing (FMEC) (2017). https://doi.org/10.1109/FMEC.2017.7946405
6. Lera, I., Guerrero, C., Juiz, C.: YAFS: a simulator for IoT scenarios in fog computing. IEEE Access 7 (2019). https://doi.org/10.1109/ACCESS.2019.2927895
7. Qayyum, T., Malik, A.W., Khan Khattak, M.A., Khalid, O., Khan, S.U.: FogNetSim++: a toolkit for modeling and simulation of distributed fog environment. IEEE Access 6 (2018). https://doi.org/10.1109/ACCESS.2018.2877696
8. Gupta, H., Dastjerdi, A.V., Ghosh, S.K., Buyya, R.: iFogSim: a toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments. Softw. Pract. Exp. 47 (2016). https://doi.org/10.1002/spe.2509
9. Nikdel, Z., Gao, B., Neville, S.W.: DockerSim: full-stack simulation of container-based software-as-a-service (SaaS) cloud deployments and environments. In: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM) (2017). https://doi.org/10.1109/PACRIM.2017.8121898
10. Mayer, R., Graser, L., Gupta, H., Saurez, E., Ramachandran, U.: EmuFog: extensible and scalable emulation of large-scale fog computing infrastructures. IEEE Fog World Congr. (FWC) (2017). https://doi.org/10.1109/FWC.2017.8368525
11. Markus, A., Kertesz, A.: A survey and taxonomy of simulation environments modelling fog computing. Simul. Model. Pract. Theory 101 (2019). https://doi.org/10.1016/j.simpat.2019.102042
12. Kecskemeti, G.: DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds. Simul. Model. Pract. Theory 58 (2015). https://doi.org/10.1016/j.simpat.2015.05.009
13. Markus, A., Kertesz, A., Kecskemeti, G.: Cost-aware IoT extension of DISSECT-CF. Future Internet 9 (2017). https://doi.org/10.3390/fi9030047
14. Markus, A., Gacsi, P., Kertesz, A.: Develop or dissipate fogs? Evaluating an IoT application in fog and cloud simulations. In: 10th International Conference on Cloud Computing and Services Science (CLOSER) (2020). https://doi.org/10.5220/0009590401930203
15. Abreu, D.P., Velasquez, K., Curado, M., Monteiro, E.: A comparative analysis of simulators for the cloud to fog continuum. Simul. Model. Pract. Theory 101 (2020). https://doi.org/10.1016/j.simpat.2019.102029
16. Google Scholar. https://scholar.google.com/. Accessed June 2020
17. iFogSim simulator. https://github.com/petergacsi/iFogSim/. Accessed September 2019
18. DISSECT-CF-Fog simulator. https://github.com/andrasmarkus/dissect-cf/tree/fog-extension/. Accessed June 2020
19. SonarCloud. https://sonarcloud.io/. Accessed June 2020
20. JProfiler. https://www.ej-technologies.com/products/jprofiler/overview.html. Accessed June 2020
21. AWS EC2 Instance Pricing. https://aws.amazon.com/ec2/pricing/on-demand/. Accessed September 2019