Flexible Representation of IoT Sensors for Cloud Simulators

Andras Markus*, Gabor Kecskemeti† and Attila Kertesz*
* Software Engineering Department, University of Szeged, Hungary
† Department of Computer Science, Liverpool John Moores University, United Kingdom
Email: g.kecskemeti@ljmu.ac.uk, keratt@inf.u-szeged.hu

***Abstract*—In Internet of Things (IoT), sensors, actuators and smart devices are connected to the Internet. Application providers combine this connectivity with novel scenarios involving cloud computing. Some require in depth analysis of the interaction between IoT devices and clouds. Research focuses on questions like how to govern such large cohort of devices (i.e., often over tens of thousands). Distributed systems simulators help in such analysis, but they are problematic to apply in this newly emerging domain. Most simulators are either too detailed (e.g., need extensive knowledge on networking), or not extensible enough to support the new scenarios. This paper introduces our attempt to show how a state of the art simulator could model generic IoT sensors. We show the fundamental properties of IoT entities represented in the simulator. Based on these properties, we present an XML based, declarative modelling language aiming at: (i) describing the behaviour of sensors and their relation to clouds, and (ii) allowing rapid prototyping of simulations. Finally, we validate the applicability of our IoT extensions in five scenarios in the field of weather forecasting.**

I. Introduction

Internet of Things (IoT) is a rapidly emerging concept where sensors, actuators and smart devices are often connected to cloud systems. Clouds are used in scenarios in which data from a large set of sensors is processed and often fed back to actuators or smart devices. As the number of devices with connected sensors and actuators are reaching new heights every day, the way they are integrated to the cloud computing ecosystem is also rapidly changing. As a result, IoT systems integrators often need new experimental techniques – e.g., simulators – which allow them to understand the behaviour of systems of previously unprecedented scale.

Recently, a wide range of IoT oriented simulators have risen [1], [2], [3]. Regrettably, these simulators are frequently limiting their use cases (e.g., big data processing). Also, often they are focused on very specific sensors, or sensor behaviour. Finally, these simulators are rarely scaling to match the number of devices foreseen in IoT systems of tomorrow.

In this paper, we lay the foundations for flexible and scalable modelling of IoT sensors through our extensions to the DISSECT-CF simulator [4]. We introduce an XML based representation for simple IoT sensor models to allow the description of basic sensor characteristics (e.g., data amount and production frequency). Building on this representation, we show how large scale simulations could be constructed and how various experiments could be done by altering sensor networks organisation or sensor data distribution. Lastly, we reveal the extension points to the currently designed system. Albeit, scaling the simulation over several nodes is a relevant topic to meet the demands of the newest IoT scenarios, this topic is out of scope here as it was discussed before by [5].

One of the earliest users of connected sensors are from the field of weather forecasting. The findings of the paper are evaluated through five weather forecasting scenarios: we used the public data available on the sensors operated by the crowdsourced meteorological service of Hungary called Idokep.hu[1]. Using the extended DISSECT-CF, we set up an extensive network of simulated sensors (with over 400 devices encompassing over 3000 individual sensors) and evaluated data collection and analysis techniques, as if they would be executed in a state of the art infrastructure as a service system.

The structure of the paper is the following. First, in Section II, we continue with the discussion of the state of the art. Next, in Section III, we discuss the extensions applied to the DISSECT-CF simulator. Later, Section IV discusses our weather forecasting case study and evaluates our extensions with a real life case. Finally, Section V concludes our work.

II. Related Work

There are many simulators available to examine distributed and specifically cloud systems. Nevertheless, there are some more specific IoT simulators closer to our approach. Han et al. [2] have designed DPWSim, which is a simulation toolkit to support the development of service-oriented and event-driven IoT applications with secure web service capabilities. Its aim is to support the OASIS standard Devices Profile for Web Services (DPWS). SimIoT [1] is derived from the SimIC simulation framework [6]. It introduces several techniques to simulate the communication between an IoT sensor and the cloud, but it is limited to compute activity modeling.

Moschakis and Karatza [7] introduce several simulation concepts for IoT systems. First, they show how the interfacing between the various cloud providers and IoT systems could be modeled (even including workload models) in a simulation. Unfortunately, they mainly discuss the behavior of cloud systems that support the processing of data originated from the IoT system. Silva et al. [8] deal with the dynamic nature of IoT systems, they investigate fault behaviors and introduce a fault model for such systems. Although faults are important,

[1]http://idokep.hu

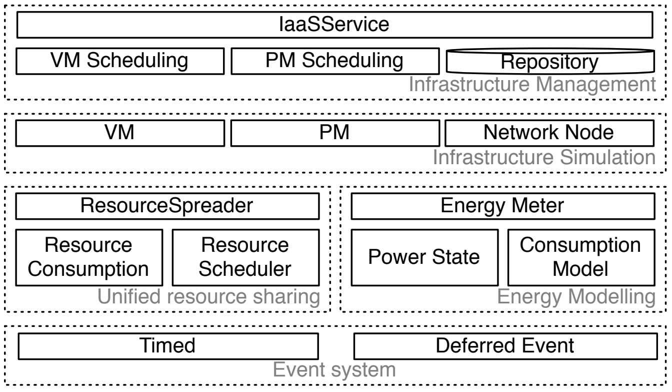

Fig. 1. The architecture of the DISSECT-CF simulator

the scalability of the introduced fault behaviors and concepts are insufficient for large scale systems.

Khan et al. [9] introduce a novel infrastructure coordination technique that supports the use of larger scale IoT systems. They build on CloudSim [10] and provide customizations that are tailored for their specific home automation scenarios and therefore limit the applicability of their extensions. Zeng et al. [3] proposed IOTSim that supports and enables simulation of big data processing in IoT systems limiting themselves to the MapReduce model. They also presented a real case study that validates the effectiveness of their simulator.

In the field of resource abstraction for IoT, efforts aimed at the description and implementation of languages and frameworks for efficient representation, annotation and processing of sensed data. The integration of IoT and clouds has been envisioned e.g. by Botta et al. [11]. They argue that system designers and operations managers face challenges to realize IoT cloud systems, due to the complexity and diversity of their requirements in terms of IoT resources consumption, customization and runtime governance. We build on these results and target our contribution at IoT Cloud simulations.

III. Our Proposed Model for IoT Sensors

We aim at supporting the simulation of thousands (or more) devices participating in previously unforeseen/existing IoT scenarios that have not been examined before in more detail (e.g., in terms of scalability, energy efficiency or management costs). As this aim requires a high performance resource sharing mechanism, we have chosen to extend the DISSECT-CF [4] simulator, because its unified resource sharing foundation.

DISSECT-CF is a compact open source[2] simulator focusing on the internals of IaaS systems. Figure 1 presents its architecture. There are five subsystems (encircled with dashed lines) implemented, each responsible for a particular functionality: (i) event system – the primary time reference; (ii) unified resource sharing – models low-level resource bottlenecks; (iii) energy modeling – for the analysis of energy-usage patterns of resources (e.g., NICs, CPUs) or their aggregations; (iv) infrastructure simulation – for physical/virtual machines and networking; and (v) infrastructure management – provides a cloud like API and cloud level scheduling.

Our extension takes into account the following IoT components: sensors, actuators and central computing services. Sensors are essential parts of IoT systems, and usually they are passive entities. Their performance is limited by their network gateway's (i.e., the device which polls for the measurements and sends them away) connectivity and maximum update frequency. Actuators are entities also limited by their network connectivity and reaction time (e.g., how long does it take to actually perform an actuation action). They also have the unique feature that allow changing the location of non-cloud entities. Finally, central computing services provide the large-scale background processing and storage capabilities needed for the IoT scenarios. According to recent advances in IoT, these services are expected to be used only if unavoidable.

Based on these generic plans we performed the extension of the DISSECT-CF simulator towards IoT. To derive the sensor models for the extension, we started by modelling a real-world IoT system. As one of the earliest examples of sensor networks are from the field of meteorology and weather forecasting, we choose to model the crowdsourced meteorological service of Hungary called Idokep.hu. It has been established in 2004, and it is one of the most popular websites on meteorology in Hungary. Since 2008 weather information can be viewed on Croatia and even on Germany. Detailed information of its system architecture and operation can also be found on the website: more than 400 stations send sensor data to their system (including temperature, humidity, barometric pressure, rainfall and wind properties), and the actual weather conditions are refreshed every 10 minutes. They also provide forecasts up to a week. They also produce and sell sensor stations capable to extend their sensor network and improve their weather predictions. These can be bought and installed at buyer specific locations. We followed a bottom-up approach to add IoT functionalities to the simulator, and implemented a weather forecasting scenario using public data available on sensors and their behaviour at http://www.idokep.hu.

Figure 2 depicts how data stored about each station in an IoT system. This description is useful to set up predefined stations from files. The tasksize attribute of `Application` (responsible for executing the scenario) defines the amount of data (in bytes) to be gathered in a cloud storage (sent by the stations) before their processing in a VM.

`Stations` have unique identifiers (i.e., a `name`) representing a weather station. We can specify their lifetime with the tag `time` by defining their `starttime` and `stoptime`. The cardinality of the supervised sensor set is set via `snumber`. Alongside the set cardinality, one can also specify the average data `size` produced by one of the sensors in the set. To set up more stations with the same properties, one can use the `count` option in the `name` tag. Data generation frequency (`freq`) could be set for the sensor set (in miliseconds). The station's caching mechanism is influenced with the tag `ratio`. This defines the amount of data to be kept at the local storage relative to the average dataset produced by the

[2]available from: https://github.com/kecskemeti/dissect-cf

```
<Application tasksize=’250000’>
<Station>
     <name count=’1’>Szeged</name>
     <freq>60000</freq>
     <snumber size=’200’>10</snumber>
     <time starttime=’500’ stoptime=’1000’>
         1000 </time>
     <maxinbw>100</maxinbw>
     <maxoutbw>100</maxoutbw>
     <storagebw>100</storagebw>
     <torepo>sztakilpdsceph</torepo>
     <storage>60000</storage>
     <ratio>1</ratio>
</Station>
</Application>
```

Fig. 2. XML-based description of IoT systems

sensors at each data generation event. If the unsent data in the local storage (which is defined in `storage`) overreaches the caching limit, the station is modelled to send the cached items to the cloud's storage (identified with its network node id specified in the `torepo` tag). The local storage is also keeping a log of previously sent data until its capacity is exceeded. The station's network connectivity to the outside world is specified by the tags `maxinbw` and `maxoutbw`, while the network capacity of the cloud storage is specified in `storagebw`. In the next section we introduce and evaluate our extensions with a weather forecasting scenario which is derived from the publicly observable real-world operation of Idokep.hu.

IV. Implementation and validation

During our implementation and evaluation, where applicable, we used publicly available information to populate our experiments. Unfortunately, some details are unpublished (e.g. sensor data sizes, data-processing times), for those, we have provided estimates and listed them below.

In the website of Idokep.hu, we learnt that currently the service operates with 487 stations. Each of them has sensors at most monitoring the following environmental properties: (i) timestamp; (ii) air and dew point temperature (in °C); (iii) humidity (%); (iv) barometic pressure (hPa); (v) rainfall (mm/hour and mm/day); (vi) wind speed (km/h); (vii) wind direction; $(viii)$ and UV-B level.

Concerning the size of such sensor data, we expect them to be save in a structured text file (eg., CSV). Stored this way, we can estimate that approximately 50 bytes (e.g., based on the website of the Murdoch University Weather Station[3]) are produced if each sensor produces data in every measurement.

In the extension the `Cloud` class can be used to specify and set up a cloud environment. This class uses DISSECT-CF's XML based cloud loader to set up a cloud environment to be used for storing and processing data from stations. This class should also be used to define Virtual Appliances modeling the application binaries doing the in cloud processing.

[3]http://wwwmet.murdoch.edu.au/downloads

The scenarios to be examined through simulations should be defined by the `Application` class. Users are expected to implement custom IoT Cloud use cases here by examining various management and processing algorithms of sensor data in VMs of a specific cloud environment. The `VmCollector` class can be used to manage such VMs, and its `VmSearch()` method can be used to check if there is a free VM available in the cloud to be utilized for a certain task. If this is not the case, the `generateAndAdd()` method can be used to deploy a new one.

Next, we detail the steps of the behaviour of our `Application` implementation which was used for all evaluation scenarios later. Step 1: Set up the cloud using an XML description. As we expect meteorological scenarios will often use private clouds, we used the model of our local private infrastructure (the LPDS Cloud of MTA SZTAKI, Hungary). Step 2: Set up the 487 stations (using a scenario specific XML description) with the previously listed 8 sensors per station. Step 3: Start the `Application` to deploy an initial VM (`generateAndAddVM()`) for processing and to start the metering process in all stations (`startStation()`). Step 4: The stations then monitor (`Metering()`), save and send (`startCommunicate()`) sensor data (to the cloud storage) according to their XML definition. Step 5: A daemon service checks regularly if the cloud repository received a scenario specific amount of data (see the `tasksize` attribute in Figure 2). If there so, then the `Application` generates tasks which will finish processing within a predefined amount of time. Step 6: Next, for each generated task, a free VM is searched (by `VmSearch()`). If a VM is found, the task and the relevant data is sent to it for processing. Step 7: In case there are no free VMs found, the daemon initiates a new VM deployment and holds back the not yet mapped tasks. Step 8: If at the end of the task assignment phase, there are still free VMs, they are all decommissioned (by `turnoffVM()`) except the last one (allowing the next rounds to start with an already available VM). Note this behaviour could be turned on/off at will. Step 9: The `Application` returns to Step 5.

A. *Evaluation with Five Scenarios*

In this sub-section, we reveal five scenarios targeting questions likely to be investigated with the help of extended DISSECT-CF [4]. Namely, our scenarios mainly focus on how resource utilization and management patterns alter based on changing sensor behaviour (e.g., how different sensor data sizes and varying number of stations and sensors affect the operation of the simulated IoT system). Note, the scope of these scenarios is solely focused on the validation of our proposed IoT extensions and thus the scenarios are mostly underdeveloped in terms of how a weather service would behave internally.

Before getting into the details, we clarify the common behaviour patterns, we used during all of the scenarios below.

[4]The scenarios are available at https://github.com/andrasmarkus/dissect-cf/tree/andrasmarkus-patch-1/experiments

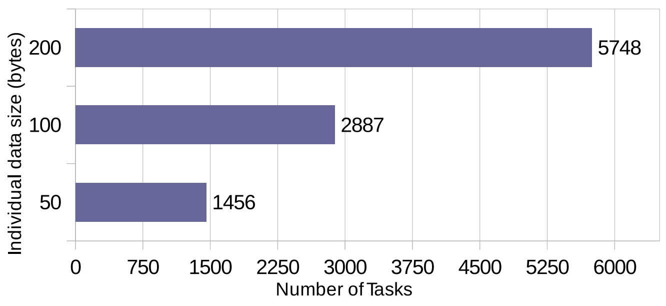

Fig. 3. Number of tasks in scenario N^{o}1

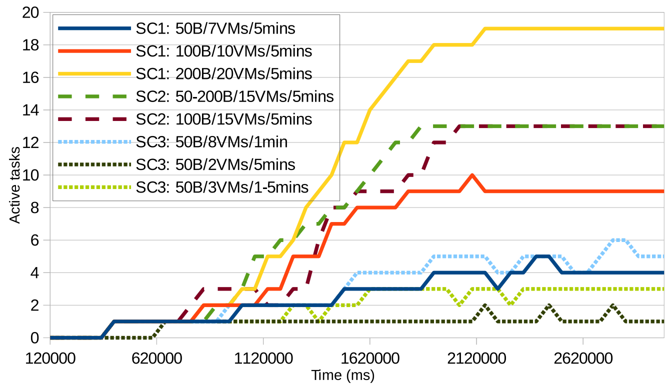

Fig. 4. The initial build-up of tasks over time in scenarios N$^o1-3$

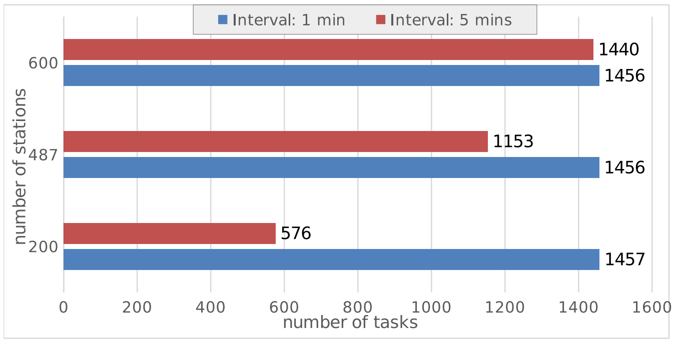

Fig. 5. Number of tasks in scenario N^{o}4

First of all, to limit simulation runtime, all of our experiments limited the station lifetimes to a single day. The start-up period of the stations were selected randomly between 0 and 20 minutes. The task creator daemon service of our `Application` implementation spawned tasks after the cloud storage received more than 250 kBs of metering data (see the `tasksize` of Figure 2). This step ensured the estimated processing time of 5 minutes/task. VMs were started for each 250 kB data set. The cloud storage was completely run empty by the daemon: the last spawned task was started with less than 250 kBs to process – scaling down its execution time. Finally, we disabled the dynamic VM decommissioning feature of the application (see step 8 in Section IV).

In scenario N^{o}1 (SC1), we varied the amount of data produced by the sensors: we set 50, 100 and 200 bytes for our cases (allowing overheads for storage, network, different data formats and secure encoding etc.). We simulated the 487 stations of the weather service. For the first case with 50 bytes of sensor data we measured 256 MBs of produced data in total, while in the second case of 100 bytes we measured 513 MBs, and in the third of 200 bytes we measured 1.02 GBs (showing linear scaling up). In the 3 cases we needed 6, 10 and 20 VMs to process all tasks respectively. The number of tasks generated by the stations can be seen in Figure 3, while the creation of tasks over time is depicted in Figure 4.

In scenario N^{o}2 (SC2), we examined the effects of varying sensor numbers and varying sensor data sizes per stations to mimic real world systems better. Therefore, we defined a fixed case using 744 stations having 7 sensors each, producing 100 bytes of sensor data per measurement, and a random case, in which we had the 744 stations with randomly sized sensor set (ranging between 6-8) and sensor data size (50, 100 or 200 bytes/sensor). The initial build-up of the tasks can be seen in Figure 4, while the total number of generated tasks were 2150 in SC2. We experienced minimal differences between the two cases, and the random case resulted in slightly more tasks.

In scenario N^{o}3 (SC3), we examined random sensor data generation frequencies. We set up 600 stations, and defined cases for two static frequencies (1 and 5 minutes), and a third case, in which we randomly set the sensing frequency between 1 and 5. In real life, the varying weather conditions may call for (or result in) such changes. In both cases, the sensors generated our previously estimated 50 bytes. The generated data in total: 316 MBs for 1 minute frequency, 63 MBs for 5 minute frequency, and 143 MBs for the randomly selected frequencies. The first case required the highest number of VMs to process the sensed data, but the randomly modified sensing frequency resulted in the highest number of tasks. The total tasks generated are 1455, and the build-up of the tasks compated to the previous scenarios are shown in Figure 4.

In the three scenarios executed so far the main application, responsible for processing the sensor data in the cloud, checked the repository for new transfers in every minute. In some cases we experienced that only small amount of data has arrived within this interval (i.e. task creation frequency). Therefore in scenario N^{o}4, we examined what happens if we widen this interval to 5 minutes. We executed three cases here with 200, 487 and 600 stations. The results can be seen in Figure 5. In Figure 6, we can read the number of VMs required for processing the tasks in the actual case. The first case has the highest difference in terms of task numbers: data coming from sensors of 200 stations needed more than 1400 tasks with 1 minute interval, while less than 600 with 5 minutes interval. It is also interesting that with 600 stations almost the same amount of tasks were generated, but with the 5 minutes interval we needed more VMs to process them.

As we model a crowdsourced service, we expect to see a more dynamic behaviour regarding stations. In the previous cases we used static number of stations per experiment, while in our final scenario, N^{o}5, we ensured station numbers

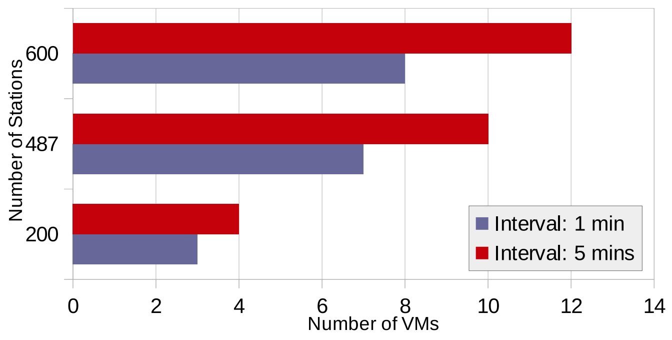

Fig. 6. Number of virtual machines in scenario N^{o}4

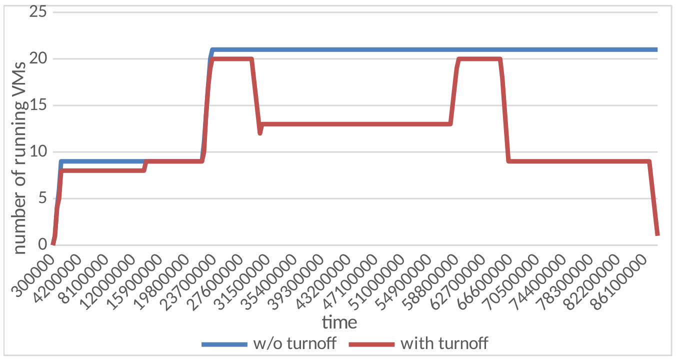

Fig. 7. Results of scenario N^{o}5

dynamically change. Such changes may occur due to station or sensor failures, or even by sensor replacement. In this scenario we performed these changes by specific hours of the day: from 0-5 am we started 200 stations, from 6-8 am we operated 500 stations, from 9 am to 15 pm we scaled them down to 300, then from 16-18 up to 500, finally the last round from 19-24 pm we set it back to 200. In this experiment we also wanted to examine the effects of VM decommissioning, therefore we executed two different cases, one with and one without turning off unused VMs. In both cases we set the `tasksize` attribute to 10 kB (instead of the usual 250kB). The results can be seen in Figure 7. We can see that without turning off the unused VMs from 6 pm we kept more than 20 VMs alive (resulting in more overprovisioning), while in the other case the number of running VMs dynamically changed to the one required by the number of tasks to be processed.

As a summary, in this section, we presented five scenarios focusing on various properties of IoT systems. We have shown that with our extended simulator, we can investigate the behaviour of these systems and contribute to the development of better solutions in this research field.

V. Conclusions

Distributed systems simulators are not generic enough to be applied in newly emerging domains, such as IoT Cloud systems. Most systems are either too detailed, or not extensible to support the to be modelled devices. Therefore in this paper we introduced a method to show how generic IoT sensors could be modelled in a state of the art cloud simulator. We showed how the fundamental properties of IoT entities can be represented in the simulator, and proposed an XML based, declarative modelling language to describe the behaviour of various sensors. We also validated our extensions in the simulator by executing 5 different scenarios of simulated IoT systems provisioning a meteorological service.

Our future work will address investigations on non-frequency based sensor data production to allow more influence on when and what kind of data is produced by a particular sensor. Also, an initial model on actuators will be proposed.

Acknowledgements

The research leading to these results has received funding from the European COST programme under Action identifier IC1304 (ACROSS), and was supported by the UNKP-UNKP-16-4 New National Excellence Program of the Ministry of Human Capacities of Hungary, and by the EU-supported Hungarian national grant GINOP-2.3.2-15-2016-00037 ”Internet of Living Things”.

References

[1] Stelios Sotiriadis, Nik Bessis, Eleana Asimakopoulou, and Navonil Mustafee. Towards simulating the internet of things. In *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*, pages 444–448. IEEE, 2014.

[2] Son N Han, Gyu Myoung Lee, Noel Crespi, Kyongwoo Heo, Nguyen Van Luong, Mihaela Brut, and Patrick Gatellier. DPWSim: A simulation toolkit for iot applications using devices profile for web services. In *IEEE World Forum on IoT (WF-IoT)*, pages 544–547. IEEE, 2014.

[3] Xuezhi Zeng, Saurabh Kumar Garg, Peter Strazdins, Prem Prakash Jayaraman, Dimitrios Georgakopoulos, and Rajiv Ranjan. IOTSim: A simulator for analysing iot applications. *Journal of Systems Arch.*, 2016.

[4] Gabor Kecskemeti. DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58P2:188–218, November 2015.

[5] Gabor Kecskemeti and Zsolt Nemeth. Foundations for simulating iot control mechanisms with a chemical analogy. In Benny Mandler, Johann Marquez-Barja, et al., editors, *Internet of Things. IoT 360° 2015, Rome, Italy, October 27-29, 2015. Revised Selected Papers, Part I*, volume 169 of *LNICST*, pages 367–376, 2016.

[6] Stelios Sotiriadis, Nik Bessis, Nikos Antonopoulos, and Ashiq Anjum. SimIC: Designing a new inter-cloud simulation platform for integrating large-scale resource management. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 90–97. IEEE, 2013.

[7] Ioannis A. Moschakis and Helen D. Karatza. Towards scheduling for internet-of-things applications on clouds: a simulated annealing approach. *Concurrency and Computation: Practice and Experience*, 27(8):1886–1899, 2015.

[8] Ivanovitch Silva, Rafael Leandro, Daniel Macedo, and Luiz Affonso Guedes. A dependability evaluation tool for the internet of things. *Computers & Electrical Engineering*, 39(7):2005–2018, 2013.

[9] Adnan M Khan, Leandro Navarro, Leila Sharifi, and Luís Veiga. Clouds of small things: Provisioning infrastructure-as-a-service from within community networks. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, pages 16–21. IEEE, 2013.

[10] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, January 2011.

[11] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. On the integration of cloud computing and internet of things. In *Int. Conf. on Future Internet of Things and Cloud*, pages 23–30. IEEE, 2014.