Acta Cybernetica **24** (2019) 83–103.

Multi-Cloud Management Strategies for Simulating IoT Applications

Andras Markus[a] and Jozsef Daniel Dombi[b]

Abstract

The Internet of Things (IoT) paradigm is closely coupled with cloud technologies, and the support for managing sensor data is one of the primary concerns of Cloud Computing. IoT-Cloud systems are widely used to manage sensors and different smart devices connected to the cloud, hence a large amount of data is generated by these things that need to be efficiently stored and processed. Simulation platforms have the advantage of enabling the investigation of complex systems without the need of purchasing and installing physical resources. In our previous work, we chose the DISSECT-CF simulator to model IoT-Cloud systems, and we also introduced provider pricing models to enable cost-aware policies for experimentation. The aim of this paper is to further extend the simulation capabilities of this tool by enabling multi-cloud resource management. In this paper we introduce four cloud selection strategies aimed to reduce application execution time and utilization costs. We detail our proposed method towards multi-cloud extension, and evaluate the defined strategies through scenarios of a meteorological application.

Keywords: cloud computing, internet of things, simulation, Pliant system

1 Introduction

In the paradigm of the Internet of Things (IoT), sensors and smart devices are connected to the Internet giving way to many opportunities to use cloud and IoT services together [1]. Since more and more devices enter the network to form IoT systems, the dataflow and the workload of the supporting services are increasing, which also raise open issues such as resource usage and cost reduction or legal compliance [5]. Hiring physical machines from virtual server parks fitting various IoT scenarios could be very expensive, and the investigation of IoT-enabled cloud service compositions is not always possible with real cloud providers. As a result, in

[a]Software Engineering Department, University of Szeged, 6720 Szeged, Hungary, E-mail: markusa@inf.u-szeged.hu

[b]Software Engineering Department, University of Szeged, 6720 Szeged, Hungary, E-mail: dombijd@inf.u-szeged.hu

DOI: 10.14232/actacyb.24.1.2019.7

many cases cloud simulators are applied to address the evaluation of such complex environments.

While network simulators could be too complex to simulate IoT and cloud systems together, due to detailed network configurations, special purpose cloud simulators may be over-tailored to cloud-specific details making it hard to express IoT needs. The number of IoT devices and usage areas are constantly growing, and some cases require immediate intervention after data processing, such as heart monitoring in smart homes, or traffic control in smart cities. This means we need new solutions and techniques for data storage, access and processing, which can be designed and evaluated in infrastructure cloud simulators extended with IoT simulation capabilities. Therefore we have chosen DISSECT-CF to perform our investigations [2].

In our earlier works we introduced IoT modeling to a traditionally cloud simulator, then combined provider pricing schemes with IoT cloud management in DISSECT-CF [3] to enable cost-aware investigations. Since cloud federations [4] provide a wider range of capabilities to users, the next step in our research was to enable the usage of multiple cloud datacenters to serve certain IoT scenarios.

In this paper we introduce four cloud selection strategies aimed to reduce application execution time and utilization costs. The default strategy uses random cloud selection for the managed IoT devices, and we also propose a load balancing and a cost minimizing strategy. Finally, for a more sophisticated strategy we apply a fuzzy-based approach. We also evaluate our proposal through scenarios derived from a real-life weather forecasting service.

The remainder of this paper is as follows: Section 2 discusses related approaches in this field, and Section 3 summarizes relevant previous works of the authors. Section 4 introduces our proposed cloud selection strategies, which are evaluated in Section 5. Finally, we conclude the paper in Section 6.

2 Related work

In the field of cloud and IoT simulations, CloudSim [6] is one of the most widespread solutions for modeling cloud system components including data centers and virtual machines, as well as investigating cloud resource provisioning policies. When IoT started to emerge, CloudSim has been extended to provide modeling capabilities for IoT system components. Khan et al. [7] proposed an infrastructure coordination technique for large scale IoT systems built on top of CloudSim. It provides customization for specific home automation scenarios, which limits the applicability of their extensions. The iFogSim [8] also extends CloudSim to simulate IoT and fog environments by measuring resource management techniques with several metrics including latency, network congestion, energy consumption and cost. They presented two case studies to demonstrate IoT modeling and resource management policies: latency-sensitive online gaming, and an intelligent surveillance application using distributed camera networks.

Besides CloudSim, we can find similar simulation approaches tailored to specific

needs. Zeng et al. [13] proposed IOTSim that supports the simulation of big data processing with the MapReduce model exemplified with a real case study. SimIoT [9] is based on the SimIC simulation framework [10], and it proposes several techniques to simulate the communication possibilities between IoT sensors and cloud components, but it is limited to compute activity modeling.

MobIoTSim [14] proposes a semi-simulated environment for investigating IoT cloud systems. It aims at mimicking the behavior of IoT sensors and devices with a mobile simulation environment. Sensor data management and system scalability can be investigated with real interconnected gateway services.

Concerning IoT management algorithms, Moschakis and Karatza [11] introduced workload models with interfacing various cloud providers and IoT systems, enabling the investigation of the behavior of cloud systems that support the processing of data originated from the IoT system. Silva et al. [12] focused on the dynamic nature of IoT systems, therefore they investigated fault behaviors with specific fault models. Unfortunately, the scalability of the introduced fault behavior concepts are insufficient for large scale systems.

Several providers offer PaaS-level cloud services with the possibility of connecting and managing IoT devices, we can find a detailed comparison of them in [15]. These solutions are usually tightly coupled for certain providers, and hide low-level details of utilization, which is an advantage for end-users, but they are not suitable for modeling low-level infrastructure operations, and developing multi-provider IoT-Cloud applications.

From these related works we can see that IoT-Cloud systems can be examined with several simulation tools, and supporting environments already exist for investigating specific behavioral methods, such as resource selection, sensor communication, big data management, energy efficiency and cost savings. Nevertheless, the combination of these aims and closer relation to real world utilization patterns still represent open issues. Our approach combines cost reduction based on real world provider pricing and multi-cloud resource selection, applied in a real-world usage scenario.

3 IoT-Cloud Simulation in DISSECT-CF

One of our main goals for choosing the DISSECT-CF cloud simulator for our investigations was its unified resource sharing mechanism. Timed events are the basic elements of this simulator, which can be recurrent time-dependent events that have a frequency value (e.g. 10 ms), which calls their methods regularly in every moment based on the given value. There are non-recurrent time-dependent events as well, that have only a delay value (e.g. 5 ms) denoting the time to be elapsed before its function has to be called. Both type of events are controlled by the inside clock of the simulator. With these build-in events we can simulate the management of IoT systems including sensors and smart devices. The configuration of IoT system properties in the simulator can be done through an XML description file. We can set the following attributes: network bandwidth, local repository size, operating time,

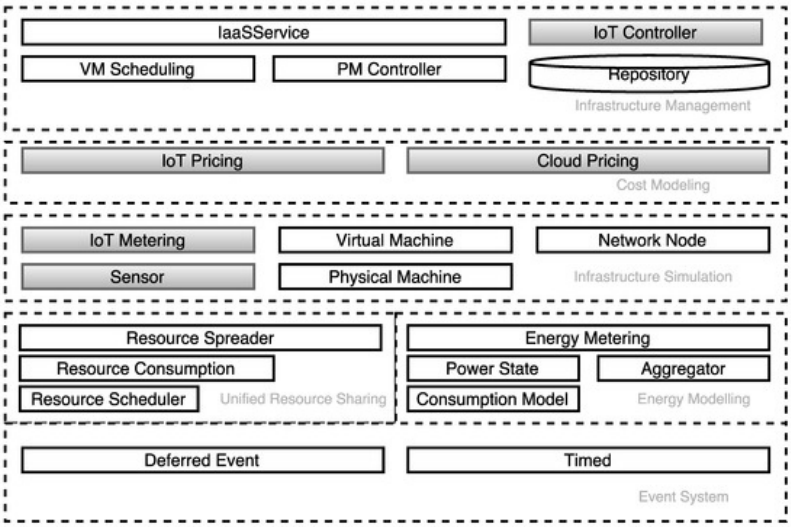

Figure 1: Architecture of the IoT-extended DISSECT-CF simulator

number of sensors and frequencies of the data generating, storing and sending, and the size of the generated files.

Two additional XML descriptions can be used to set provider pricing properties. Usually the cloud side pricing is used to calculate the costs of virtual machines (VMs) used to run an IoT application. It defines a fixed monthly cost per VM instance, but some providers charge the hour per price for every instance an application needs. To manage data coming from IoT devices and sensors, we need to calculate the IoT side costs, that can also be set based on real provider pricing schemes (e.g. Amazon, Azure, IBM and Oracle). In general, the IoT prices are calculated after the generated data traffic in MB following the "pay as you go" approach, while some providers charge after the number of messages exchanged in a month, or set a monthly device per price or messages sent in a day. All these three XML description formats and possible parameters are presented and discussed in our previous work [3].

In general, a simulation is performed by executing the following steps: first, a cloud is set up using an XML description (we used the model of a Hungarian private cloud infrastructure called the LPDS Cloud of MTA SZTAKI [24]), then the necessary amount of stations are initialized and the VM parameters are loaded from additional XML files, which also describe the cloud and IoT costs. Next, the IoT application is started with the deployment of an initial VM in the defined cloud, followed by the start of metering and data generation processes of device stations. IoT and cloud operations are continuously monitored to calculate the resource consumption costs. During execution, a broker service checks if the cloud repository received a scenario-specific amount of data, if so, then a compute task

will be generated and deployed in a VM for data processing.

Finally, sensor data generation, compute task creation and execution are repeated till the end of the simulation, with possible starting and stopping of VMs. At the end of the simulation we can retrieve information about end user costs concerning the utilization of IoT and cloud resource consumption.

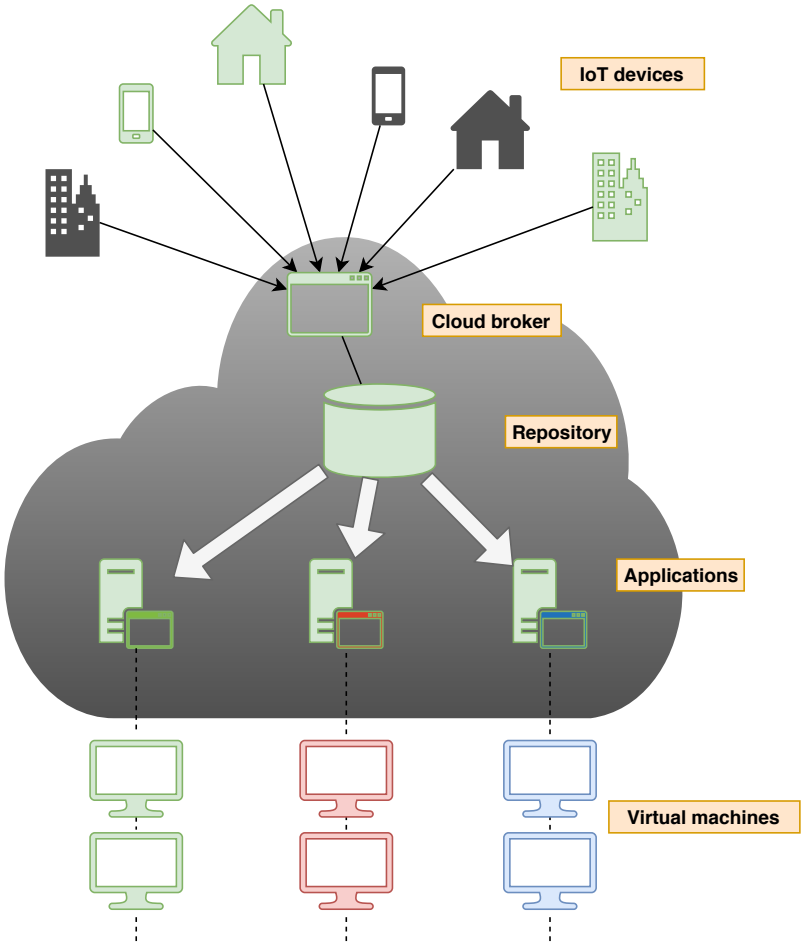

Figure 2: Application execution in the extended DISSECT-CF simulator

4 The proposed cloud selection strategies

The main research question of this paper is how we can influence the behavior of an IoT application, if the sensors can have different allocation strategies for multiple clouds. In the earlier version of the extended DISSECT-CF we could exploit only

one cloud datacenter to start VMs, therefore all sensors and smart devices were connected to this specific cloud, and all the generated data of the sensors were processed by virtual machines running in the same cloud (as we summarized in the previous section). In this single cloud setup, a cloud can have a preloaded cost calculation policy with a single pricing scheme. Smart devices usually have different sensors and usage frequencies affecting data generation methods that can influence cloud service operation and also the provider pricing. As a result, a single cloud could be easily overloaded, and the unprocessed data could hinder the operation of the IoT application causing longer response times, even service unavailability in real-world services.

The formerly added components of the simulator by our previous work towards IoT extension can be seen in Figure 1 denoted by grey background. Our current contribution targets the top levels of the simulator architecture, and aims to enable the use of multiple IoT Controllers and pricing schemes.

In this work we introduce the possibility of multi-cloud management for IoT cloud simulations in DISSECT-CF. During the start of the simulation we can set up different clouds using extended XML descriptions denoting sets of physical machines and repositories with various properties. Another improvement is the introduction of a cloud broker, which can manage different VM queues. These queues may have virtual machines with different pricing policies, and within a simulation the broker can decide, to which cloud (and to which VM queue) the IoT devices should be connected, thus where the generated data should be sent and processed in an application. This revised IoT Cloud management architecture is depicted in Figure 2, showing one cloud with three different applications mapped to three different VM queues. These extensions make the simulator more flexible and capable of performing scalability experiments involving multiple cloud providers.

In order to enable easy and repeatable system configuration, we defined a new XML format to configure VM flavors with prices for the applications. An example of this XML structure can be seen in Figure 3, which defines two flavors. With this flavor model we can specify the required VM resources (*cpu-cores, ram*), the cost of the VM (*price-per-tick*), the number of boot instructions affecting the boot time (*startup-process*), the network traffic (*network-load*) and the local disk size requirement (*req-disk*) of the new instance. Flavors can be identified by the *name* attribute, and they must be unique in the configuration. In this example we defined two different clouds (4 CPU cores with 4 GB RAM, and 2 CPU cores with 2 GB RAM), which allocate almost 10 GBs of the local disc.

An example XML with two application descriptions is presented in Figure 4. We defined a daemon service frequency (*freq*) to regularly check the repository for unprocessed data. The *tasksize* attribute tells the highest amount of unprocessed data that can packaged in one compute task to be executed by virtual machines. The selected computing infrastructure is identified by the *cloud* tag, and finally, the VM flavor to be used for executing the compute tasks can be specified in the *instance* tag (by referring to the *name* attribute of the flavor model).

In this example both defined applications use the formerly defined flavors and two different clouds (identified by 'Cloud1' and 'Cloud2' unique name). The fre-

```
<?xml version="1.0"?>
<flavors>
   <flavor name="amazon-large">
     <ram>4294967296</ram>
     <cpu-cores>4</cpu-cores>
     <price-per-tick>0.000015</price-per-tick>
     <core-processing-power>0.001</core-processing-power>
     <startup-process>100</startup-process>
     <network-load>0</network-load>
     <req-disk>10000000000</req-disk>
   </flavor>
   <flavor name="azure-small">
     <ram>2147483648</ram>
     <cpu-cores>2</cpu-cores>
     <price-per-tick>0.000001</price-per-tick>
     <core-processing-power>0.001</core-processing-power>
     <startup-process>100</startup-process>
     <network-load>0</network-load>
     <req-disk>10000000000</req-disk>
   </flavor>
</flavors>
```

Figure 3: Sample description using the flavor XML model

quency value defines that the daemon service should repeat the virtual machine handling functions (generate, shutdown and reboot the VM based on the actual load of unprocessed data) every 5 minutes.

In the IoT paradigm the sensors are passive entities of the systems, thus their performance is limited by the operation frequency (i.e., data generation, storing, transfer to the cloud), up-time and network connection. Usually, large amounts of sensor data are sent from the smart devices to cloud resources for further computation and analysis. Since resource consumption can be costly, IoT application owners can reduce their expenses by selecting a provider having a suitable pricing scheme.

In this paper we defined four different strategies to perform cloud provider selection (to be done during each IoT device (or sensor) start-up), which can be denoted by setting the strategy field of the XML description of each device participating in the simulation. The strategy for a smart device is defined in the device XML description format presented in Figure 5 with the *strategy* tag. Also the network settings (*maxinbw, maxoutbw, diskbw*) of the local repository (*reposize*) are set with data caching function (*data-ratio*). We can configure the life time of

```
<?xml version="1.0"?>
<applications>
   <application tasksize="2500000">
      <name>Weather-1</name>
      <freq>300000</freq>
      <cloud>cloud1</cloud>
      <instance>azure-small</instance>
   </application>
   <application tasksize="2500000">
      <name>Weather-2</name>
      <freq>400000</freq>
      <cloud>cloud2</cloud>
      <instance>amazon-large</instance>
   </application>
</applications>
```

Figure 4: Sample description using the application XML model

the device (*starttime, stoptime*), the number of sensors it has (*sensor*), the size of the generated data (*filesize*) and the generation and sending frequency (*freq*). The device settings can be applied for a group of devices using the *number* attribute. In this example (Figure 5) the configuration file defines 487 devices running for 6 hours and each device generates 50 bytes of data by its 8 sensors. Detailed XML samples and schemes can be found in [20].

We propose four different strategies for multi-cloud management: (i) *random*, (ii) *cost-aware*, (iii) *runtime-aware* and (iv) *Pliant*. In the next subsections we introduce these strategies.

4.1 Basic strategies

With the *random* strategy the cloud broker chooses one of the available applications running in the simulated clouds randomly for an actual IoT device (sensor or station).

The *cost-aware* strategy looks for the cheapest available VM in a cloud (based on their static pricing properties), thus it compares the prices of the required VM flavors for a given device. Its algorithm orders the VMs by their price-per-tick value. This solution may be more suitable for IoT applications having relatively small data processing needs or less susceptible for the processing time, because cloud providers usually offer lower resource capacities for less costs.

In the *runtime-aware* strategy, the corresponding algorithm ranks the available VMs (residing in different clouds) by a specific value defined by the ratio of the

```
<?xml version="1.0" encoding="UTF-8"?>
<devices>
   <device starttime="0" stoptime="21600000"
           number="487" filesize="50">
      <name>test1</name>
      <freq>60000</freq>
      <sensor>8</sensor>
      <maxinbw>1000</maxinbw>
      <maxoutbw>1000</maxoutbw>
      <diskbw>1000</diskbw>
      <reposize>60000</reposize>
      <data-ratio>1</data-ratio>
      <strategy>random</strategy>
   </device>
</devices>
```

Figure 5: Sample description using the device XML model

number of already connected devices and the number of the available physical machines of the hosting cloud. This is a dynamic strategy taking into account the actual load of the available clouds. Applications having longer data processing needs may prefer this strategy.

4.2 The Pliant strategy

Fuzzy sets were introduced in 1965 with the aim of reconciling mathematical modeling and human knowledge in the engineering sciences. Fuzzy logic means that we can not decide whether the value is true or not. The true lies between the true and false value. Fuzzy logic offers a very valuable flexibility for reasoning [17]. Most of the building blocks of the theory of fuzzy sets were proposed by Zadeh, especially fuzzy extensions of classical basic mathematical notions like logical connectives, rules, relations and quantifiers. Over the last century, fuzzy sets and fuzzy logic [16] have become more popular areas for research, and they are being applied in fields such as computer science, mathematics and engineering. This has led to a truly enormous literature, where there are presently over thirty thousand published papers dealing with fuzzy logic, and several hundreds books have appeared on the various facets of the theory and the methodology. However, there is not a single, superior fuzzy logic or fuzzy reasoning method available, although there are numerous competing theories.

The Pliant system is a kind of fuzzy theory that is similar to a fuzzy system [18]. The difference between the two systems lies in the choice of operators. In fuzzy theory the membership function plays an important role, but the exact definition

of this function is often unclear. In Pliant systems we use a so-called distending function, which represents a soft inequality. In the Pliant system the various operators, which are called the conjunction, disjunction and aggregative operators, are closely related to each other. In the Pliant system we have a generator function and using this function we can create aggregation operator, conjunctive operator or disjunctive operator. In the Pliant systems the corresponding aggregative operators of the strict t-norm and strict t-conorm are equivalent, and DeMorgans law is obeyed with the corresponding strong negation of the strict t-norm or t-conorm.

The Pliant system has a strict, monotonously increasing t-norm and t-conorm, and the following expression is valid for the generator function:

$$f_c(x)f_d(x) = 1, \tag{1}$$

where $f_c(x)$ and $f_d(x)$ are the generator functions for the conjunctive and disjunctive logical operators, respectively. This system is defined in the [0,1] interval.

The operators of the Pliant system are

$$c(\mathbf{x}) = \frac{1}{1 + \left(\sum_{i=1}^{n} w_i \left(\frac{1-x_i}{x_i} \right)^{\alpha} \right)^{1/\alpha}} \tag{2}$$

$$d(\mathbf{x}) = \frac{1}{1 + \left(\sum_{i=1}^{n} w_i \left(\frac{1-x_i}{x_i} \right)^{-\alpha} \right)^{-1/\alpha}} \tag{3}$$

$$a_{\nu_*}(\mathbf{x}) = \frac{1}{1 + \left(\frac{1-\nu_*}{\nu_*} \right) \prod_{i=1}^{n} \left(\frac{1-x_i}{x_i} \frac{1-\nu_*}{\nu_*} \right)^{w_i}} \tag{4}$$

$$n(x) = \frac{1}{1 + \left(\frac{1-\nu_*}{\nu_*} \right)^2 \frac{x}{1-x}}, \tag{5}$$

$$\kappa_{\nu}^{(\lambda)}(x) = \frac{1}{1 + \frac{1-\nu_0}{\nu_0} \left(\frac{\nu}{1-\nu} \frac{1-x}{x} \right)^{\lambda}}$$

where $\nu_* \in]0, 1[$, with generator functions

$$f_c(x) = \left(\frac{1-x}{x} \right)^{\alpha} \qquad f_d(x) = \left(\frac{1-x}{x} \right)^{-\alpha}, \tag{6}$$

where $\alpha > 0$.

The operators c, d and n fulfill the DeMorgan identity for all ν, a and n fulfill the self-DeMorgan identity for all ν, and the aggregative operator is distributive with the strict t-norm or t-conorm. The ν value express the expected value of the given context. This means that if the given x value is greater than ν, then the operators increase the value of x. The opposite is true when x is smaller than ν.

Table 1: Normalization parameters

Parameter	Lambda	Shift
General VM cost	-1.0/96.0	15
Cost of the application	-1.0	(maxPrice-minPrice)/2
Workload	-1.0	maxWorkload
Number of VM	-1.0/8.0	3
Number of stations	-0.125	sumStations/appSize
Number of active stations	-0.125	sumStation/activeStation
VM memory size	1.0/256	350
VM CPU	1.0/32	3

These algorithms calculate a score for each cloud using the environment properties. The calculation step includes a normalization step, where we apply the Sigmoid function. In the normalization step it should be mentioned that if the normalized value is close to one, it means it is a more valuable property, and if the normalized value is close to zero, it means it is a less prioritized property. For example, if the CPU utilization of the VM is high, the normalization algorithm should give a value close to zero.

In a previous work [19], we used the Pliant system approach to schedule applications to VMs in a cloud by minimizing energy consumption. There we experienced that uncertainty could be well tolerated with this approach, and better results can be achieved with this model than traditional approaches.

In this work we create a new algorithm that can predict, which cloud could be the best for managing a given IoT device. This algorithm is also based on the Pliant logic, therefore for each cloud (i.e. for each VM queue in a cloud) it calculates a score number. The first step of the algorithm is to normalize the data into the [0,1] interval. We apply a Sigmoid function for this purpose. We define the following properties for each cloud VM: general VM cost, current cost of application, workload, number of running VMs in the hosting cloud, number of devices that are already connected to a cloud, memory size and number of CPUs. In Table 1 we can see the exact values of the normalization functions.

After the normalization step we modify the normalized value to emphasize the importance of the result. This means that if the given x value is greater than our expectation (ν) than we will increase the value of x. the opposite is true when the given x is smaller than ν. To achieve this we will modify the normalized value by using the Kappa function shown in Figure 6 with $\nu = 0.4$ and $\lambda = 3.0$ parameters:

$$\kappa_\nu^\lambda(x) = \frac{1}{1 + \left(\frac{\nu}{1-\nu}\frac{1-x}{x}\right)^\lambda} \tag{7}$$

Finally, to calculate a cloud score number for the given application. For this

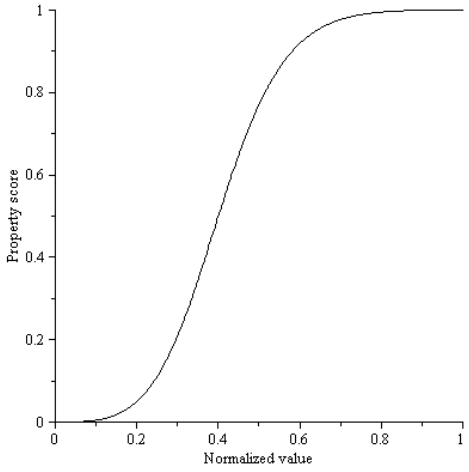

Figure 6: The Kappa function

manner we can use conjunction, disjunction or aggregation operator. The conjunctive operator is similar as the and operator. This means that if one of the value is small, then the result will be also small. The opposite is true for disjunctive operator, that is similar to or operator. If on of the value is large, the result will be also large. The aggregation operator lies between the disjunctive or conjunctive operator, that is why we use this operator:

$$a_{\nu,\nu_0}(x_1,\cdots,x_n)=\frac{1}{1+\frac{1-\nu_0}{\nu_0}\frac{\nu}{1-\nu}\prod_{i=1}^{n}\frac{1-x_i}{x_i}}, \tag{8}$$

where ν is the neutral value and ν_0 is the threshold value of the corresponding negation. Here we don't want to threshold the result so both parameters have the same value 0.5. The result of the calculation is always a real number that lies in the [0,1] interval. So we calculate the score for all clouds (i.e. VM queues of clouds) to find which one is the most suitable for a given device.

5 Evaluation with weather forecasting scenarios

One of the earliest examples of sensor networks comes from the field of weather prediction, therefore we chose to model meteorological services based on available public information. Not only the managing architecture, but the generated sensor data is also modeled, which are in most cases: temperature, humidity, barometric pressure, rainfall and wind properties. In our model the weather conditions are regularly refreshed by the service websites in every 5 minutes, but the sensors are able to generate data in every minute, which needs caching not to overload the service. In this paper our proposed algorithms address the optimization of cloud

Table 2: Detailed Bluemix, Azure and Amazon pricing-based private cloud configurations used in the evaluations

Cloud	Bluemix		
Flavor	Small	Medium	Large
Hourly price (Euro)	0.0378	0.149	0.295
CPU (Cores)/RAM (GB)	1/1	4/2	8/4
Cloud	Azure		
Flavor	Small	Medium	Large
Hourly price (Euro)	0.019	0.0579	0.297
CPU (Cores)/RAM (GB)	1/1.75	2/3.5	8/14
Cloud	Amazon		
Flavor	Small	Medium	Large
Hourly price (Euro)	0.0229	0.0415	0.3327
CPU (Cores)/RAM (GB)	1/2	2/4	8/32

Table 3: Detailed multi-cloud configuration for the evaluations

Cloud	Physical machines
LPDS-1	1 PM - 32 cores, 128 GB RAM 4 PMs - 8 cores, 12 GB RAM
LPDS-2	1 PM - 64 cores, 128 GB RAM 1 PMs - 48 cores, 128 GB RAM 1 PMs - 32 cores, 128 GB RAM 9 PMs - 8 cores, 12 GB RAM
LPDS-3	2 PM - 64 cores, 128 GB RAM 2 PMs - 48 cores, 128 GB RAM 2 PMs - 32 cores, 128 GB RAM 18 PMs - 8 cores, 12 GB RAM

side costs with enhanced allocation of the stations (i.e. devices). Which means we can define more cloud providers with their own pricing schemes, but we use only one IoT provider (therefore the IoT side cost cannot be optimized).

5.1 Scenario N^{o}1

In the first scenario we chose to model the crowd-sourced meteorological service of Hungary called *Idokep.hu* [21]. In this scenario we aimed to model its real-world operation: all stations have 8 sensors (represented by a device in our model), the message size of the sensors can be set up to 0.05 KBs, and the sensors generate

Table 4: Evaluation results of the scenario N^{o}1

Strategies	Cost-aware	Random	Runtime-Aware	Pliant
App-1 cost	0	1.119	1.119	1.119
App-2 cost	0	2.027	2.027	2.027
App-3 cost	0	16.167	16.223	16.223
App-4 cost	0	1.842	1.842	1.842
App-5 cost	0	7.300	7.300	7.300
App-6 cost	0	14.426	14.426	14.426
App-7 cost	1.769	0.974	0.972	0.974
App-8 cost	0	2.827	2.822	2.82 7
App-9 cost	0	14.478	14.454	14.478
Total cost (Euro)	1.769	61.164	61.188	61.219
No. of used VMs	5	9	9	9
Total tasks	227	2619	2616	2619
Timeout (min)	1.76	4.01	4.05	2.06

data in every minute. The start-up period of the stations were selected randomly between 0 and 20 min. In order to exemplify the usage of different cloud selection strategies, we defined periodic start-up and shut-down dates for certain stations (e.g. to represent malfunctions or failures). We simulate a whole day of operation (from 0:00 a.m. to 24:00 a.m.), and we start the simulation by setting up 200 stations at 0:00 a.m. At 2:00 a.m. we start 100 more, and at 10:00 a.m. 200 more to scale the total number of operated stations up to 500. At 2.00 p.m. we shut down 200 stations to scale down the number of running station to 300 by 10 p.m. At the end of the day the total number of running stations return to 200. This means the total number of operated meteorological stations in this scenario are 500 (which denotes a relatively small scale, nation-wide system).

With these station management timings we run four different test cases: (i) all stations use the random strategy, (ii) all stations use the cost-aware strategy, then (iii) all stations use the runtime-aware strategy. Finally, (iv) we used the Pliant strategy in the last experiment. For this evaluation we configured three clouds based on the *LPDS-1* cloud description from Table 3, and every cloud can run application instances (to execute compute tasks) in three VM flavors defined in Table 2 (that makes 9 possible application instances in total).

We executed the formerly defined scenario with the four test cases. The results of the experiments can be seen in Table 4. After executing this scenario the applications processed 173.75 MBs of data. The so-called timeout parameter denotes how much time it took for the application to terminate (i.e. to perform all remaining data processing operations) after the last station stopped working (at 24:00 a.m.). As we can see from these results, the cheapest solution is the cost-aware strategy (1.769 Euros) with these simulation parameters, it also has the shortest

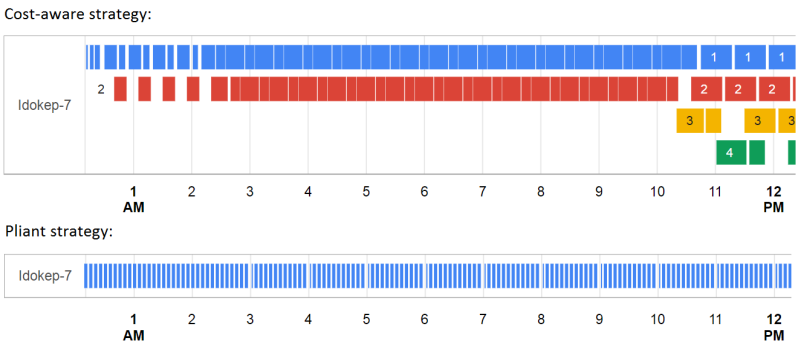

Figure 7: Timeline comparing task allocations of pliant and cost-aware strategies in Scenario N^{o}1

timeout (1.76 minutes) and it utilized the least virtual machines. This strategy only used 5 instances of the cheapest VM while the other strategies used 1 VM instance in every running application. Since the stronger virtual machines (having higher costs) processed the tasks faster than the weaker ones (as expected), they had to generate tasks more frequently. In general, choosing the cheapest VM for an application may result in serious delay in real time systems, but in this case our simulated system can operate with weaker resources in real time due to the small amount of sensor data to be processed. The beginning of the simulation there is no virtual machine running, which can serve any task execution request, thus each simulation has to wait at most 5 minutes to deploy one and allocate tasks, which means all timeout values of the strategies are acceptable.

The random, runtime-aware and Pliant strategies use unnecessarily more expensive virtual machines, which results in more than 60 Euros of cost in every case, but we can see the advantage of the pliant strategy: it tries to minimize the timeout value. In this case, it achieved a timeout of 2.06 minutes, which is the second best result. It shows that our Pliant strategy focused more on execution time reduction than cost savings.

Figure 7 shows the allocated tasks of an application running in the simulation with the pliant and cost-aware strategies for the first 12 hours. Every box denotes a different task and boxes having the same color were processed by the same virtual machine. The lengths of the tasks refers to their execution time. We can see that the tasks of the cost-aware strategy processed relatively medium amount of data resulting in many, not too narrow boxes on the timeline. In case the amount of unprocessed data was growing, the system started to scale up the number of utilized virtual machines. Meanwhile the pliant strategy worked with stronger and faster virtual machines, which resulted also in many tasks with small amount of data, but using only the same, best fit VM. This explains the difference between the number of total tasks of these strategies. Next we investigate a scenario of a higher scale.

Table 5: Evaluation results of the scenario N^{o}2

Strategies	Cost-aware	Random	Runtime-Aware	Pliant
App-1 cost	0	3.563	3.544	3.542
App-2 cost	0	3.745	3.707	3.721
App-3 cost	0	12.396	12.451	12.451
App-4 cost	0	5.799	5.796	5.783
App-5 cost	0	9.384	8.324	8.748
App-6 cost	0	12.157	12.132	12.034
App-7 cost	26.419	3.061	3.063	3.090
App-8 cost	0	5.156	5.243	5.166
App-9 cost	0	11.261	11.187	11.112
Total cost (Euro)	26.419	66.527	65.451	65.651
No. of used VMs	109	180	170	173
Total tasks	1722	1830	1819	1838
Timeout (min)	631	86	86	71

5.2 Scenario N^{o}2

In the second scenario we aimed to simulate a larger, world-wide system. An international meteorological service called OpenWeatherMap [22] is operated by the Openweather IT company [23], which was established in 2014 by a group of experts in Big Data and image processing. As their website suggests, they manage over 40000 meteorological stations all over the world. Our goal with this scenario is to investigate how IoT applications behave in such large-scale environments. Similarly to the first scenario, we used three clouds configured with Amazon, Azure and IBM Bluemix cloud provider pricing defined in Table 2, but we modified the physical parameters of the simulated private clouds (to be able to cope with the higher number of stations) as defined by *LPDS-2* in Table 3. The number of running weather stations has been increased to 40000, each of them works with 8 sensors and generate 50 bytes of data every minute. We run this scenario to simulate 6 working hours. In the beginning we started 10000 stations, then we added 10000 stations more in the next hours to reach 40000 stations by the fourth hour.

The results of the second scenario is shown in Table 5. After executing this scenario the applications processed 4.008 GBs of data. In the previous scenario the cost-aware strategy was good choice both cost and runtime, but problems may occur for systems with higher scale. In this case the cheapest schedule was provided by the cost-aware strategy with 26.419 Euros, but it had 631 minutes ($\sim$10.51 hours) timeout, which is almost twice longer than the simulated working time (i.e. 6 hours). For applications which are not sensitive to low latency, the cost-aware strategy can be still an acceptable opportunity to decrease costs, but for time-dependent applications (e.g. smart systems, weather forecasting systems) other strategies are

Table 6: Evaluation results of the scenario N^{o}3

Strategies	Cost-aware	Random	Runtime-Aware	Pliant
App-1 cost	0	3.512	3.552	3.552
App-2 cost	0	3.724	3.731	3.804
App-3 cost	0	13.283	12.479	12.451
App-4 cost	0	6.233	5.830	5.824
App-5 cost	0	9.197	8.523	8.386
App-6 cost	0	12.428	12.157	11.960
App-7 cost	26.489	3.085	3.071	3.070
App-8 cost	0	5.224	5.185	5.195
App-9 cost	0	11.904	12.251	12.152
Total cost (Euro)	26.489	66.429	66.784	66.397
Used VMs	172	183	184	185
Total tasks	1722	1905	1889	1893
Timeout (min)	526	36	36	36

needed. Nevertheless, the cost-aware strategy utilized the lowest number of VMs, too. The random and the runtime-aware strategies have the same timeout (with 86 minutes), but the runtime-aware approach operated with less virtual machines (with 10 VMs) and saved around one Euro compared to the random one. The pliant strategy was even better with almost the same price, since it reached the most favorable timeout (with 71 minutes).

5.3 Scenario N^{o}3

In the third scenario we configured our private cloud to be the strongest, having twice as many resources as in the second scenario (detailed in the *LPDS-3* parameter setup of Table 3), while the rest of the configuration (the applications and the stations) remained untouched, thus the final amount of generated (and processed) data was the same as in the previous scenario.

Table 6 shows the results of the third scenario. With the increased physical resources the running time have decreased, but the cost-aware strategy still required 526 minutes (∼8.76 hours) timeout, after the last station stopped working.

If we take a look at the figures, we can see that most strategies benefited from the stronger clouds: they all managed to reduce the timeout significantly. The cost-aware strategy remained the cheapest one, but the number of used virtual machines increased the most against the other strategies compared with the second scenario. The amount of unprocessed data grew faster, than the number of available virtual machines, thus when the application operated with the maximum number of stations the stronger resources could provide more virtual machines to reduce timeout. Comparing the other three strategies, it shows minimal deviation in the

Table 7: Evaluation results of the scenario N^{o}4

Strategies	Cost-aware	Runtime-Aware	Pliant
Total cost (Euro)	10.442	41.765	38.84
Used VMs	81	51	51
Total tasks	685	1384	1242
Timeout (min)	41	31	24.9

used virtual machines or the costs. The pliant approach uses the most virtual machines (185), but it was the cheapest with 66.397 Euros, but all strategies has the same timeout value with 36 minutes. This means that by increasing the number of resources, the strategies behave differently.

5.4 Scenario N^{o}4

In the previous scenarios the station allocation strategies had to choose only 2-4 times to select VM-queues for the applications processing sensor data of the stations. One of advantages of the pliant approach is that it is able to take into account more features of the underlying systems, but for this strategy these scenarios were too static, having only a small number of decision points. Thus in the last, fourth scenario we defined a more dynamic scenario, where we managed 11500 stations in the following way. Every half an hour, 500 stations were started to operate and the whole simulation run for 12 hours. The pliant algorithm had to decide more often than in the former cases. Our aim with this scenario is to prove that this sophisticated algorithm is able to decrease both the costs and the runtime at the same time. The results can be seen in Figure 7. The processed data for the whole experiment is 1.54 GBs. For this scenario we used a different cloud setup as well. We configured three clouds based on the *LPDS-1*, *LPDS-2* and *LPDS-3* cloud description of Table 3, respectively.

As expected the cheapest solution is the Cost-aware algorithm with 10.442 Euros, which also has the highest timeout with 41 minutes. This strategy used the highest number of virtual machines, which is also a disadvantage, if the cloud provider calculates the cost based on the number of VMs. Comparing the other strategies (here we neglected the random approach), the pliant and the runtime-aware strategies used the same number of virtual machines, but the Pliant algorithm managed to reduce both the cost and the runtime most effectively.

6 Conclusion

Cloud Computing solutions act as supporting services for the IoT world. Applications in this newly emerged field are continuously growing, and further research is

still needed to resolve open issues, to optimize system management and to reduce utilization costs for both providers and end-users.

In this paper we introduced four cloud selection strategies aimed to reduce IoT application execution time and usage costs. We evaluated these strategies through scenarios derived from a real-life weather forecasting service. The results have shown that we can achieve significant cost savings or makespan reduction in multi-cloud systems by using one of our proposed strategies.

Our investigations showed that if the components of IoT-Cloud systems (including sensors, smart devices and virtual resources) change often, a static scheduling and placement algorithm (ignoring the actual load, the type of virtual machines and the number of physical resources) can provide increased latencies and costs. Our presented a dynamic approach based on the Pliant method can adapt to the actual state of the underlying, possibly multi-cloud systems, therefore it can find better placement of devices resulting in lower costs and response times.

Our future work will address IoT scenarios from other smart domains (e.g. smart farming), as well as the modeling of additional sensor and device types. We also plan to extend our cloud selections algorithms to minimize IoT side costs, and to introduce energy-aware algorithms for smart device management. We also plan to perform experiments with models of recent infrastructures (e.g. Agrodat [25], MTA Cloud [26] or the Cloud for Education [27]).

The presented scenarios and the source code of the IoT extended DISSECT-CF with the mentioned XML description formats and the XML Schema Document files are available at [20].

7 Acknowledgement

This work was supported by the Hungarian Government under the grant number EFOP-3.6.1-16-2016-00008.

References

[1] A. Botta, W. de Donato, V. Persico, and A. Pescape. Integration of Cloud computing and Internet of Things. Future Gener. Comput. Syst. 56, pp. 684-700, 2016.

[2] G. Kecskemeti. DISSECT-CF: A simulator to foster energy-aware scheduling in infrastructure clouds. *Simul. Model. Pract. Theory*, 58P2, 2015.

[3] A. Markus, A. Kertesz, G. Kecskemeti. Cost-aware iot extension of dissect-cf. *Future Internet*, 9(3), 2017.

[4] A. Kertesz. Characterizing cloud federation approaches. In: Cloud computing: challenges, limitations and R&D solutions. Computer communications and networks. Springer, Cham, pp. 277-296, 2014.

[5] G. Gultekin Varkonyi, Sz. Varadi, A. Kertesz. Legal Aspects of Operating IoT Applications in the Fog. In: Fog and Edge Computing: Principles and Paradigms. John Wiley & Sons, Hoboken, pp. 411-432, 2019.

[6] Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.; Buyya, R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, *41*, 23–50, 2011.

[7] Khan, A.M.; Navarro, L.; Sharifi, L.; Veiga, L. Clouds of small things: Provisioning infrastructure-as-a-service from within community networks. Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on. IEEE, pp. 16–21, 2013.

[8] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, R. Buyya. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. Softw. Pract. Exper. 47:12751296, 2017.

[9] Sotiriadis, S.; Bessis, N.; Asimakopoulou, E.; Mustafee, N. Towards simulating the Internet of Things. 28th International Conference on Advanced Information Networking and Applications Workshops (WAINA), pp. 444–448, 2014.

[10] Sotiriadis, S.; Bessis, N.; Antonopoulos, N.; Anjum, A. SimIC: Designing a new Inter-Cloud Simulation platform for integrating large-scale resource management. IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), pp. 90–97, 2013.

[11] Moschakis, I.A.; Karatza, H.D. Towards scheduling for Internet-of-Things applications on clouds: a simulated annealing approach. *Concurrency and Computation: Practice and Experience*, *27*, 1886–1899, 2015.

[12] Silva, I.; Leandro, R.; Macedo, D.; Guedes, L.A. A dependability evaluation tool for the Internet of Things. *Computers & Electrical Engineering*, *39*, 2005–2018, 2013.

[13] Zeng, X.; Garg, S.K.; Strazdins, P.; Jayaraman, P.P.; Georgakopoulos, D.; Ranjan, R. IOTSim: A simulator for analysing IoT applications. *Journal of Systems Arch.*, 2016.

[14] A. Kertesz, T. Pflanzner, T. Gyimothy. A Mobile IoT Device Simulator for IoT-Fog-Cloud Systems. Journal of Grid Computing, 2018. 10.1007/s10723-018-9468-9.

[15] T. Pflanzner, A. Kertesz. A Taxonomy and Survey of IoT Cloud Applications. EAI Endorsed Transactions on Internet of Things, 2018.

[16] J. Dombi. A general class of fuzzy operators, the de morgan class of fuzzy operators and fuzziness measures induced by fuzzy operators. Fuzzy Sets and Systems 8, 1982.

[17] Yager, Ronald R. and Zadeh, Lotfi A. An Introduction to Fuzzy Logic Applications in Intelligent Systems Kluwer Academic Publishers, 0792391918, 1992

[18] J. Dombi. Pliant system. IEEE International Conference on Intelligent Engineering System Proceedings, Budapest, Hungary, 1997.

[19] A. Kertesz, J. D. Dombi, A. Benyi. A Pliant-based Virtual Machine Scheduling Solution to Improve the Energy Efficiency of IaaS Clouds. Journal of Grid Computing, Vol. 14, pp. 41–53, 2016.

[20] DISSECT-CF extensions towards IoT. Online: https://github.com/andrasmarkus/dissect-cf/tree/pricing/. Accessed in September, 2018.

[21] Websize of Idokep.hu. Online: http://idokep.hu. Accessed in August, 2017.

[22] OpenWeatherMap Station information site. Online: https://openweathermap.org/stations-old. Accessed in September, 2018.

[23] OpenWeather company website. Online: https://openweather.co.uk/about. Accessed in September, 2018.

[24] LPDS Cloud of MTA SZTAKI. Online: https://www.sztaki.hu/tudomany/reszlegek/lpds. Accessed in September, 2018.

[25] R. Lovas, K. Koplanyi, G. Elo. Agrodat: A Knowledge Centre and Decision Support System for Precision Farming Based on IoT and Big Data Technologies. Special theme: Smart Farming, ERCIM News 113. April, 2018.

[26] MTA Cloud website. Online: https://cloud.mta.hu. Accessed in January, 2019.

[27] Cloud for Education website of KIFU. Online: http://kifu.gov.hu/szolgaltatasok/ ikt/felho/cloud_for_education. Accessed in January, 2019.