Self-adaptive Device Management for the IoT Using Constraint Solving

Ghada Moualla, Sébastien Bolle, Marc Douet, Eric Rutten

Self-adaptive Device Management for the IoT Using Constraint Solving

Ghada Moualla, Sebastien Bolle, Marc Douet
Orange Labs
38 Meylan, France
Email: Firstname.Lastname@orange.com

Eric Rutten
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG,
F-38000 Grenoble France,
Email: Eric.Rutten@inria.fr

***Abstract*—In the context of IoT (Internet of Things), Device Management (DM), i.e., remote administration of IoT devices, becomes essential to keep them connected, updated and secure, thus increasing their lifespan through firmware and configuration updates and security patches. Legacy DM solutions are adequate when dealing with home devices (such as Television set-top boxes) but need to be extended to adapt to new IoT requirements. Indeed, their manual operation by system administrators requires advanced knowledge and skills. Further, the static DM platform — a component above IoT platforms that offers advanced features such as campaign updates / massive operation management — is unable to scale and adapt to IoT dynamicity. To cope with this, this work, performed in an industrial context at Orange, proposes a self-adaptive architecture with runtime horizontal scaling of DM servers, with an autonomic Auto-Scaling Manager, integrating in the loop constraint programming for decision-making, validated with a meaningful industrial use-case.**

I. Introduction

With Device Management (DM), an operator or a service provider is able to remotely manage connected devices deployed at the customer's premises. The main DM features are [1]: (*i*) Provisioning: which targets device initial and in-life configuration, (*ii*) Monitoring: which allows detecting anomalies such as malfunctioning devices using log traces and data collection, (*iii*) Maintenance: which allows firmware and configuration updates, and (*iv*) Troubleshooting: which is remote actions to fix service and device errors.

Currently, DM solutions are widely deployed and mainly used for Smart Home service management. However, IoT platforms with DM features and standards have been developed to incorporate DM features such as firmware and configuration update, for two reasons : one is to accommodate the expansion of the Internet of Things (IoT) that offers a wide range of new smart applications [2] (e.g., Smart City, Smart Building, Smart Industry), the other is for environmental reasons of sustainability (e.g., device reuse, and lifespan enhancement).

While connectivity and cloud analytics are considered essential aspects of an IoT architecture, one of the most critical is the management of IoT [3]. This will help register connected devices, bring them online efficiently, ensure that they work properly and securely after being deployed, and send configuration or firmware updates remotely. However, with a very large number of IoT devices distributed across one or more geographic locations, monitoring and maintaining these devices can be an overwhelming task if done at the individual physical level. Moreover, the conventional centralized DM approach becomes a serious limitation.

To face these limitations, this work is performed in an industrial context at Orange, and proposes an approach to self-adaptive Device Management for the IoT using constraint solving, validated on an industrial use-case. We introduce a new IoT DM architecture based on an autonomic manager, called Auto Scaling Manager (ASM). We have leveraged the MAPE-K (Monitor-Analyze-Plan-Execute over a shared Knowledge) Autonomic Computing reference architecture [4] to build this manager that is able to manage the DM system and adapt at runtime the required number of DM servers to handle the evolution of both the IoT device fleet and the physical infrastructure. Furthermore, a constraint programming model [5] is integrated into this manager and used for decision-making on the placement of DM service within the infrastructure.

The adaptive solutions for the scaling and placement of distributed components is a well-known topic in the context of distributed systems and the Constraint Programming (CP) has also used for a variety of real-world optimisation problems including placement problem. However, our main contribution involves a Constraint Programming-based autonomic loop approach in the context of DM IoT. In which, the system information is gathered and analyzed at runtime, in order to dynamically revise and adapt the constraint optimization criteria.

We evaluate experimentally the feasibility of our approach considering a privacy use-case. The objective is to analyze the ASM behavior in terms of adaptation decisions and to show how it scales horizontally, with respect to the evolution of both the DM system and the physical infrastructure. Our contributions are:

- An autonomic DM architecture for IoT devices to handle the device fleet evolution at runtime. For that, a Constraint Programming paradigm is integrated into an autonomic feedback loop.
- An experimental validation of the architecture for a privacy scenario.

The paper is structured as follows. Section II covers the background and the related work for the good understanding of our work. Section III states the targeted problem. Section IV details our proposed architecture. Section V details our

experimental evaluation and results. The last section concludes our paper and raises some future perspectives.

II. State of the Art

A. Device Management in IoT Context

DM operations are performed remotely via a management server, managed by a DM operator, that sends out the management operations to the management clients, hosted on the managed devices to ensure their proper functionality. The server and clients communication is based on dedicated protocols, e.g., TR-069 [6], OMA Lightweight Machine to Machine (*LWM2M*) [7].

DM of IoT is vital to maintain the proper functionality of IoT devices, keep them secure, and ensure the evolution and maintenance of their growing number, while anticipating new demands for devices and services. Compared to the legacy DM, new challenges are faced [8], [9]: (i) Heterogeneity: moving from a limited set of device types to a wide variety of devices [2], (ii) Dynamicity: moving from rather stable internal states, i.e., battery and network conditions, towards versatile states[10], (iii) Privacy: many IoT services leverage sensor data to adapt to the local context. It is critical to adapt these services to protect end users privacy and customers confidential and personal information [11], and (iv) Scalability: moving from a few devices managed by centralized systems to a massive number of devices that needs a distributed management.

In the works [12], [13], authors proposed: (i) automating of DM operations, and (ii) adjusting the execution speed of the campaign firmware update to device and infrastructure capabilities, i.e., hardware, current load and network congestion to address IoT heterogeneity and dynamism. Their adaptation strategy is based on operational measures, e.g., the error rate of DM operations and the infrastructure response time. Our work takes a further step towards scalability and privacy management by distributing DM operations close to the customer's premises based on Edge Computing.

Cloud computing consists in delivering computing and storage services over the Internet. It offers the advantages of flexibility and the ability to store/analyze data. However, when cloud computing is used for IoT, new challenges emerge.

With the huge number of heterogeneous IoT devices, IT operational bandwidth (BW) consumption is significant, especially from managing device provisioning, commissioning, decommissioning, and ensuring firmware upgrades. Further, as more devices appear, there is a need for a scalable DM solution to adapt to varied deployment scenarios and enable seamless integration and management of these distributed devices. We believe that Edge Computing, with all its features, has a promising potential to ensure end-users privacy (among other benefits as security and latency) when performing DM operations, compared to a centralized DM platform hosted in the cloud.

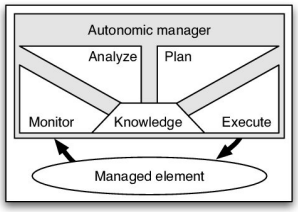


Fig. 1: MAPE-K Autonomic Loop, problem in reference [4]

B. Autonomic Computing

Autonomic Computing (*AC*) is defined as the self-management capabilities of a system [14] to respond to the increasing system administration complexity. It has proven to be effective in minimizing administrator involvement in the computer systems management, where an autonomic system is able to adapt to both external and internal changes by re-configuring itself (Fig. 1 shows the AC reference architecture).

Due to the huge number of heterogeneous devices involved in IoT system, the manual management and maintenance is impractical. In [15], the authors advocated that to address the problems of manual management, automatic approaches is needed. In their study, they stated why autonomic computing is useful and how to use it in the IoT context. Furthermore, automated architectures for IoT DM have been proposed in [13], [12] for automated device targeting (i.e., defining the appropriate devices for a DM operation) and for detecting errors/anomalies before generalizing patching to all target devices in a given fleet).

C. Service Placement in Fog and Edge using Constraints

Given the massive number of IoT devices, along with their related applications, the applications deployment in the Fog/Edge is needed to cope with the latency and privacy requirements [16]. This placement problem has been addressed in the literature [16], [17], [18] with several solutions based on different application scenarios, e.g., monolithic applications, network assumptions, e.g., infinite link bandwidth, and objective functions e.g., latency, cost. These solutions can be classified as follows: (i) Exact solution using Integer Linear Programming (ILP) [19]/ Constraint Programming [20], (ii) Approximations, and (iii) Heuristics [21] and Meta-heuristic, e.g., Genetic Algorithm (GA).

The placement solutions, that mainly rely on ILP or on heuristic approaches, are not generic enough to deal with the features of all applications [20], as they are: (i) uneasily extensible to incorporate new application / infrastructure features or to integrate new placement constraints and (ii) non-upgradable to exploit any user-implied resolution approach.

CP has these following appealing features: (i) It provides a generic and easy-to-upgrade service placement, (ii) It provides a faster solution even in a reasonably large scale environment (in [20] it is compared to and outperformed other algorithms such as ILP and GA with 1200 nodes), and (iii) The CP code is small and easy to implement. Thus, CP seems attractive to

adopt it for DM problem (More details in Sec. V). Many open-source CP solvers are available, such as the Choco solver [22] and OR-tools [23] and other commercial solvers, such as IBM CPLEX CP Optimizer [24].

For solving the Service Placement Problem (SPP), different optimization strategies are proposed in the literature [18], [16]. We distinguish two categories: (i) mono-objective that optimizes only one objective function, and (ii) multi-objective optimization that optimizes simultaneously many objective functions. These optimization functions are: (i) Latency: for delay-sensitive applications, (ii) Resource utilization: such as minimizing used bandwidth, (iii) Cost: There are two types of costs: the networking cost for data transmission charges and associated expenses, and other expenses related to storage, migration, and so on, (iv) Energy consumption: it includes when the service is sent by the end-user to the Edge device, when the Edge node processes the service; and when the Edge needs the Cloud, and (v) Other metrics: such as congestion ratio, i.e., ratio between the service links requirements and the physical links capacity.

D. State of the Art Synthesis

Device management is a fundamental issue, especially when it comes to the IoT environment, with its emerging challenges. To cope with these challenges, Autonomic Computing with its self-management capabilities, appears as an attractive solution for managing the DM system. Further, leveraging the benefits of the Edge Computing paradigm, we aim to provide the DM operations for IoT clients in a distributed manner to meet both the clients' requirements and the providers' concerns.

To this end, relying on Edge Computing together with Autonomic Computing MAPE-K architecture, we provide an autonomic manager with the ability for autonomous horizontal scaling in terms of the number of DM servers to adapt to the evolution of both the DM device fleet and the physical infrastructure. Learning from the literature and moving a step further, we rely on Constraint Programming to model and solve at runtime the DM placement problem by integrating a CP solver into the autonomic loop of our proposed autonomic manager. Moreover, to the best of our knowledge, our work is the first work that tackle the DM of IoT devices in a distributed manner.

III. Problem Definition

Deploying and managing large fleet of IoT devices is challenging due to their geographical distribution [25] and it can be an overwhelming task if done at the individual physical level. Ensuring the up-to-date state of this fleet is perturbed by fleet composition variations or by the availability of new firmware or configuration. The variation depends on two types of events: Device arrival which corresponds to the first connection of a customer's newly acquired object that usually comes with an outdated factory-installed firmware; and Device departure that occurs when an owner unsubscribes to an operator's offer, requiring a configuration to reset the device to factory settings or when a device shuts down their network access to save energy for instance.

As a result, there is a strong and growing need for an autonomous system to manage the device fleet. This feature can considerably reduce the time and effort required to sustain the health and the performance of devices throughout their lifecycle. The automation of the DM system was considered in works [12], [13] which gave us a starting point for our work.

Moreover, when it comes to firmware updates, e.g., security patches or other updates, called over-the-air (OTA) updates, the conventional centralized DM approach becomes a serious limitation since designing a DM system to handle hundreds of devices is entirely different from designing one to handle billions. The consumption of IT bandwidth resulting from managing these devices will be significant. Therefore, an elastic DM system is very essential to ease and ensure secure, fast and proper batch updates for device fleet while removing the pressure on both the nodes that host the DM components and the links that are used for server-clients communication by scaling well to manage this huge device fleet.

Knowing that, the Edge computing architecture, which brings essential data processing capabilities close to the network edge, provides a compelling solution for IoT DM use case that addresses the following issues.

- Save Bandwidth: DM operations do not have to be sent over long routes between the server hosted in a data center and the end devices. With Edge Computing, the DM operations could be provided through the distributed servers via different Edge nodes at different locations near to the end users;
- Guarantee the privacy: Management of IoT end devices locally (e.g., gateway at the client geographical place) or in the proximity of the end devices could ensure the user's privacy. Given this privacy need, a DM server can be restricted to be deployed on specific areas (zone), namely Dedicated Zones (DZs). For example, a factory requiring device operations to be performed inside its LAN (Local Area Network);
- Support devices mobility: Edge computing is better suited to support end-user mobility than the centralized DM platform and to enable the seamless firmware updates management while ensuring the required latency.

IV. Proposed DM Architecture

The proposed architecture for self-adaptive IoT DM is depicted in Fig. 2. It allows the dynamic horizontal scaling of the distributed DM servers based on designated policies (e.g., physical resource utilization thresholds, Privacy requirement).

It has two main layers: (i) Control layer which is comprised of a centralized autonomic manager called ASM and a system administrator that represents an external manager-to force external actions when needed or to give high-level objectives - and (ii) Infrastructure layer that involves the physical nodes/links that hosts the Administrative layer, i.e., DM servers, and the IoT devices.

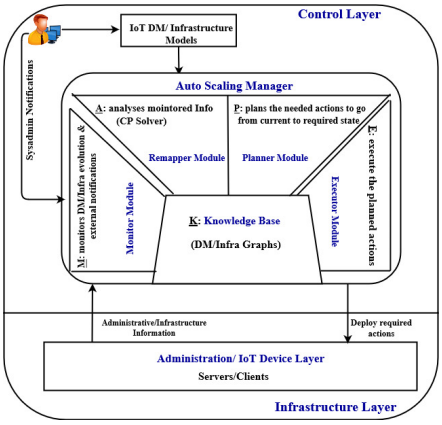
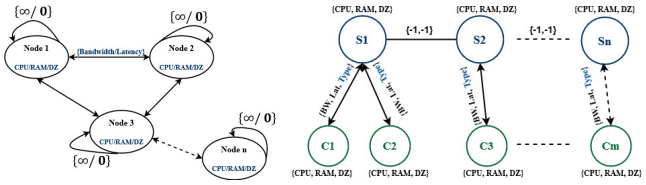


Fig. 2: Self-adaptive IoT DM Architecture based on MAPE-K reference



(a) Physical Infrastructure (b) IoT DM System

Fig. 3: Input Graph Models

This architecture is based on two main models, depicted as graphs, which represent the DM system and the underlying infrastructure (more details in Sec. V). On the one hand, the infrastructure model (Fig. 3(a)) is dedicated to the specification of the underlying topology that will host the DM service components. On the other hand, the DM system model (Fig. 3(b)) is intended to the representation of the DM system entities along with their requirements (i.e., servers and clients).

The lifecycle associated with this architecture involves these two models that will evolve over time, e.g., arrival/departure of DM clients, failures in infrastructure nodes/links. These models are used as the input of a specific constraint program integrated into the (Re)Mapper module of the ASM. In the Sec. IV-A, we explain ASM in more details.

A. Auto Scaling Manager (ASM)

This autonomic manager is built following the MAPE-K architecture and is responsible for managing of the DM system from its design till the end of the system's lifecycle. Thus, the objective is to specify a particular DM system /Infrastructure topology to be modeled, by a system administrator or DM service provider, and then handled at runtime. This implies making horizontal scaling decisions of the DM servers to adapt to any new condition dynamically, i.e., adding/removing one or more DM servers at runtime. This work is realized through the following modules:

- Monitor module: is responsible of observing the evolution of both DM system and infrastructure and forwarding the information to the Remapper module for analysis;
- (Re)Mapper module: analyses the information from the system administrator or the Monitor to decide on the target state of the DM system and then it solves the DM system placement within infrastructure (See Sec. V) ;
- Planner module: Based on the Mapper information, it extracts and selects the required actions to move to the required state of DM system (e.g., add/remove DM server(s)/client(s));
- Executor module: is responsible of deploying the required actions planned by the Planner module.

Finally, the Knowledge Base contains graph models of current and past infrastructure and DM service configurations.

B. ASM Work Methodology

The autonomic feature of the ASM is based on MAPE-K loop (in Fig. 2). It has three inputs: DM system/ Infrastructure models and the selected objective defined manually by the System administrator at initial step, named *Design time*. The infrastructure/DM graph models and the initial placement solution for DM system, along with the fore-coming graphs and placement solutions, will be stored in the ASM's Knowledge base. The output of the ASM is the DM servers that needs to be launched/stopped on particular infrastructure nodes along with their binding with their clients.

These graph models serve as an input to a CP solver that is integrated into the analyzer step, i.e., (A in Fig. 2). In this work, we consider the constraint satisfaction programming for solving the DM placement problem which provides at design time a feasible placement of the DM system within current infrastructure. However, this module is generic in that different placement approaches could be selected for both mapping/remapping steps.

Thereafter, the System evolution will be gathered by the Monitor which involves- based on the adopted scenario: (i) changes in the underlying infrastructure (e.g., node arrival/departure, resources loads), (ii) changes in the DM system (e.g., clients arrival/ departure), and (iii) other system administrator notifications, such as updating the solver objective to accommodate an urgent security campaign firmware updates instead of "normal" firmware/ configuration updates (M in Fig. 2).

The monitored information is passed to the analysis step, whenever a new change is observed. The analyser takes into account the last system state stored in its knowledge base. As a result, a new graph(s)- that reflects the new requirements and changes- will be generated and the constraint solver will be called to decide on the mapping of the new system's elements. Thus, the (Re)mapper module will be called first at the design step and then whenever a new information is observed and transmitted by the Monitor.

After receiving this information, the planner (P in Fig. 2)) produces a set of actions to be applied to move from the current system state to the target one by extracting the difference between the current and the target mapping, to do only necessary actions (e.g., decide the required number of server/client to be started/stopped). Finally, the Execution step applies these actions (E in Fig. 2)).

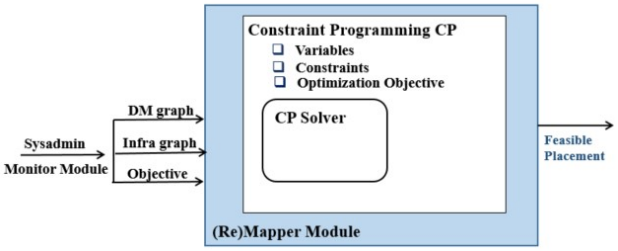


Fig. 4: Integrating CP in the ASM MAPE-K architecture

This whole process, after the design step, will be repeated in a loop based on the monitored information and the runtime conditions imposed by the Administrator. We further show in Fig. 4 how the constraint programming is integrated within ASM (Re)mapper.

Finally, in the case that the monitor detects a reconfiguration need and a CP solver is called, we may obtain that no feasible placement solution is found, for example, because there are no enough resources in the infrastructure while satisfying all required constraints. Thus, the Remapper module might need to wait for some time, so that other services release some infrastructure resources or some clients leave to try again with the solver. In such case, the ASM will keep the last solution and a notification will be sent to the system administrator.

C. Supported Use-Cases

The need for DM system reconfiguration is based on the targeted evolution type. Different changes may result in a need for a new system configuration in the DM context. Privacy is an important concern that is adopted in this work [11], where a client may need to be managed by a specific DM server or a DM server placed in a specific area due to some privacy considerations (e.g., the IoT device(s) belongs to a private enterprise or for a VIP (Very Important Person)). Thus, whenever new IoT devices ask to join the system and get registered and managed by a DM server, these newcomers will need some resources (e.g., CPU (Central Processing Unit) and RAM (Random Access Memory)) along with a specific privacy constraint. Further, IoT devices might leave the system so we need to stop the related DM clients, which results in reducing the pressure on the currently running servers.

Any DM system evolution at runtime observed by the Monitor (through an external notification issued by the administrator) will be reported to and analyzed by the Remapper. In case new devices arrival, ASM will individually check their privacy requirements to see if they can be registered with one of the existing servers. If the current servers cannot satisfy the new devices, a new server(s) need to be added the DM system, so a new DM graph will be generated to reflect the new system and the constraint solver will be called to decide on placement of the new graph.

While privacy is considered in our work to trigger the scaling process, but it is just a use-case, as there are other situations that might require adaptation and that can be handled by our proposed architecture. For example, adding more DM servers to adapt to the increased number of IoT devices communicating with a single server, and then balancing the load among these servers instances represent a new motive for scaling to avoid the stress on one server and efficiently manage the huge device fleet. In such scenario, any information leading to redeployment/instantiation need will generate a new DM graph and launch again the CP solver.

The inform storm represents a real life use-case when a huge amount of IoT devices reboot at one time, e.g., after a power breakdown. Another use-case is the smart vehicles [26] which consists of numerous sensors and actuators that automate various tasks, such as traffic monitoring and braking. In this scenario, it is necessary to keep the vehicle's firmware up-to-date, and to provide a convenient solution to the problem of installing a new firmware that could be released to fix bugs or provide new functionality. A significant challenge with this use-case is the mobility, where a vehicle may disconnect from a certain DM server placed on a specific Edge node and require to reconnect to an alternate DM server according to its current geographic location. By automatically scaling new DM servers and placing them close to the moving vehicle, this mobility problem can be handled.

V. DM servers Placement Problem

The mathematical formulation of the placement problem given in [20] is strongly aligned with our DM service placement problem. Therefore, we have benefited from this model and modified the following parts to comply with our needs. In the first update, we have changed the definition of the proposed locality constraint (See Sec. V-B). Instead of enforcing some service components to be placed on specific physical nodes, we have introduced the dedicated zones to which each physical node belongs. Then, each DM component will have a privacy requirement regarding the accepted zone of the hosting node.

Another variation is the optimization objective. Here, we have proposed a multi-objective optimization to decide at runtime which objective we need to optimize (more details in Sec. V-C). Further, the CP solver is integrated in a feedback loop in order to make adaptation decisions at runtime.

A. System Model

1) Infrastructure model: It is defined as a directed graph G^I= <H, ε>, where H is the physical nodes (e.g., Edge and IoT devices) and ε= $H \times H$ is the network links between the nodes : $\forall h \in H; \exists\, U(h)\, where\, U(h) = (CPU, RAM, DZ)$

In the infrastructure graph, we defined for each node its available CPU and RAM resources in addition to its own dedicated zone. For the physical links, we define the available bandwidth and latency as follows:

$\forall e_{i,j} \in \epsilon; \exists LAT(e_{i,j}) and BW(e_{i,j})$, where LAT($e_{i,i} = 0$) and BW($e_{i,i} = \infty$) (In future work, the link type can be added as an additional characteristic).

2) DM System Model: We consider that our DM system represents as a service and could be modeled as a graph $G^a =< C, L >$, where C is the components of our DM

system and $L = C \times C$ is the communication links between these components.

The nodes of this graph represent the DM servers and clients along with their resources requirement, namely CPU, RAM, and privacy :

$\forall c \in C; \exists\ u(c)$, where $u(c) = (Rcpu, Rram, Rdz)$ The $(Rcpu, Rram)$ are the resource requirements of each DM components and Rdz is the privacy requirement where each DM system's components need to be placed at some geographical place (e.g., customer premises for privacy issue).

In addition, for the links needed for the communication between the servers and the clients, we define the following requirements: $\forall k \in L;\ \exists\ Reqlat(k)\ and\ Reqbw(k)$ which is the latency and the bandwidth requirements of the DM system communication links.

3) Placement: After modeling both the physical infrastructure and the DM systems as graphs, the next step is to decide where to place all DM system's components and the communication links on the infrastructure physical nodes and links, respectively.

The placement solution must satisfy all the resources requirements of the DM components (namely, CPU, RAM, BW and Latency) while, at the same time, respecting the underlying infrastructure available resources. Based on methods provided in the related work we decide to use the constrained programming approach for solving the placement problem (this choice was justified in the Sec. II).

B. Problem Formulation with Constraint Programming

Here we present the formulation model for the DM placement problem. Inspired by the related work [20], for the technical needs of the modeling (to ensure we have a full connected graph), the infrastructure graph is increased by adding a super-sink node (α), with unlimited resources (CPU, RAM...), to which all graph nodes can access. The links that connect the infrastructure nodes to α, and α to itself, have an infinite capacity. In the following, the variables related to both nodes and links are declared. $\forall k \in L : s = \{s_k \mid k \in [1, |L|]\}$ where s_k is the physical node that hosts the source component of a DM system link (k), where $(s_k \in H)$.

$t = \{t_k \mid k \in [1, |L|]\}$ where t_k is the physical node that hosts the target component of a link (k), where $(t_k \in H)$.

$n = \{n_{k,j} \mid k \in [1, |L|], j \in [1, |H|]\}$ where $(n_{(k,j)} \in H)$ is the physical node at position (j) in the path of link (k).

$h = \{h_i \mid i \in [1, |C|]\}$ where $(h_i \in H)$ is the physical node that hosts a component (i).

$p = \{p_k \mid k \in [1, |L|]\}$ where $(p_k \in H)$ is the position of the target component(t_k) in the path (n_k).

$a = \{a_{k,j} \mid k \in [1, |L|], j \in [1, |H|]\}$ represents a physical link between $(n_{k,j})$ and $(n_{k,j+1})$ in the path of a DM system's link (k), where $(a_{k,j} \in \varepsilon)$.

$b = \{b_{k,j} \mid k \in [1, |L|], j \in [1, |H|]\}$ represents the bandwidth of the physical link $(a_{k,j})$.

Finally, l represents the physical link latency $(a_{k,j})$, where $l = \{l_{k,j} \mid k \in [1, |L|], j \in [1, |H|]\}$

After defining our problem model variables, we present the necessary constraints to be applied to control the possible combinations of values that these variables could obtain.

Constraint 1: *BINPACKING* constraints (knapsack-based reasoning [27]). Here we ensure that the sum of all mapped components demands does not exceed the maximum available CPU and RAM capacities of the infrastructure nodes,.

$BINPACKING(\langle h, Reqcpu\rangle, CPU)$

$BINPACKING(\langle h, Reqram\rangle, RAM)$

Constraint 2: To fix a DM system's component (i) to a specific location or DZ, we use this locality constraint:

$DZ(h_i) = Loc(i), \forall\, i \in C$

Constraint 3: Node at position (0) in the path (n_k) hosts the source component of a DM communication link (k).

$n_{k,0} = s_k$

Constraint 4: Node at position (p_k) in the path (n_k) hosts the target component of a DM communication link (k).

$n_{k,p_k} = t_k$

Constraint 5: When the source and the sink components are the same, they will be hosted on the same physical node.

$s_k = t_k \leftrightarrow p_k = 1$

Constraint 6: To avoid cycles in a path (n_k), the *ALLDIFFERENT* filtering algorithm [28] is used to prevent similar values for the variable $(n_{(k,j)})$.

$ALLDIFFERENT(n_{k,j}, \forall j \in \{1, ..., |H|\})$

Constraint 7: Any path (n_k) ends with at least one occurrence of α (the super-sink added node to the infrastructure graph). For that, the *REGULAR* constraint [29] is added to ensure that the corresponding sequence of values taken by variables belong to a given regular language.

$REGULAR(n_k, "[\hat{}\alpha] + [\alpha] + ")$

Constraint 8: When two DM communication links $(k, k^{'})$ share a same component, then the physical node that hosts the target component of the first link will be the same node the hosts the source component of the second link. $t_k = s_{k'}$

Constraint 9: This constraint is a BINPACKING constraint to respect the bandwidth limit of each physical link.

$BINPACKING(\langle b, Reqbw\rangle, BW)$

Constraint 10: These last two constraints concern respecting the latency of each DM communication link, and the end-to-end latency along the path of the DM communication link, respectively:

$l_{k,j} = LAT(a_{k,j}), \forall\, k \in L, \forall\, j \in \{0, ..., |n_k|\}$

$\sum_{j=0}^{|n_k|} l_{k,j} \leq Reqlat(k), \forall\, k \in L$

Moreover, in the DM context, the communication link type is an important requirement to be considered when solving the placement problem, where various communication technologies between devices are used such as Wi-Fi, Bluetooth, and ZigBee (based on the IoT device). Thus, when solving the placement problem, the link type could be added to our Infrastructure/DM models to better fit the DM context.

C. DM Optimization Function

Different objective functions have been introduced in the literature (see Sec. II for more details) of placement problem. However, the question that comes to mind is that: **What are**

TABLE I: Objective Optimization Classification

Metric	Description
Resources	Minimize the number of used Nodes
	Minimize the total used links' BW
Load balancing	Maximize each physical node minimum remaining CPU cores
	Maximize each physical link minimum remaining BW
Latency	Minimize the latency for urgent security patches

the optimization metrics to consider in the case of DM service provided to IoT users? To answer this, we provide a simple classification for placement objectives metrics that could fit for the DM context in the Table I. The optimal management of infrastructure resources is an important metric. With the objective *Resource*, we aim to minimize the computational or network resources used. For example, the DM system involves communications between servers and their bulk clients and communication traffic will pass through network links, that are shared with other services/applications, so minimizing network consumption is important for the DM context. However, this goal could imply that some nodes/links are overloaded while others are underloaded.

With the *Load balancing* objective, we aim to reduce the stress on both physical nodes and links (i.e., congestion rate) by balancing the load between the participant nodes and links in our infrastructure by choosing the least loaded nodes and links in the infrastructure. Finally, with the *Latency* objective, given that one of DM's features is to send security patch updates, it becomes necessary to ensure the quick arrival of such updates. To ensure this, we need to minimize the latency between the clients and their own servers.

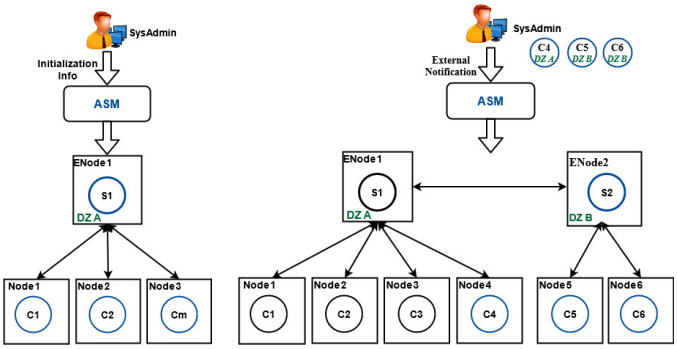
From the discussion above and given the DM system features, there is no single objective function to be considered statically. Rather, it appears that this function depends on the chosen scenario and the actual communication information exchanged between the servers and their clients. To do this, a multi objective function is introduced with *coefficients* variables (α, β, ..., etc.) that will be activated/deactivated at runtime by getting the value of 1 or 0, respectively. We formulate the final placement objective as follows:

$Objective = \alpha\ Obj1 + \beta\ Obj2 + ...$

For our proof of concept and experimental validation we choose between these objectives: (i) Minimizing the total number of participating nodes, and (ii) Minimizing the total consumed bandwidth. Our optimization objectives are formulated as follow: $Obj1 = Min \sum_{i=0}^{|H|} h_i$, where $(h_i \in H)$ is the physical node that hosts a component (i) ; $Obj2 = Min \sum_{j=0}^{|n_k|} b_{k,j}$, where $b_{k,j} = BW(a_{k,j}), \forall a_{i,j} \in \epsilon$.

VI. Experimental Validation

We present here our experimental setup to assess our approach's ability to automatically adapt by horizontal scaling the DM servers number based on both the DM system and the infrastructure's evolution. First, we present our use case and then detail the technical architecture of our setup. Then, we present our environment and ASM implementation.



(a) Design time (b) Run Time

Fig. 5: Privacy Use Case, DM System Evolution

A. Target Use Case

To assess our architecture, we considered the challenge of preserving the privacy of users who use a set of smart objects connected to each other and, potentially, to other users' objects. In such a case, IoT objects can use and propagate a lot of information about the features they offer and provide sensitive information about the users that they do not intend to reveal, e.g., knowing the features of most adopted objects by users. In this case, users may require their devices to be managed by an authorised server or by a server located in an authorised geographical place.

Therefore, we consider the following use case. Initially (See Fig. 5(a)), the DM system administrator will start a DM server in a specific location in the infrastructure and let the DM clients register to it as it meets their privacy requirements (i.e., based on the server locality). From here, the ASM will be in charge of managing the DM system throughout its life cycle.

After some time, new devices will arrive and ask to join the fleet and register with one of the servers respecting their privacy constraints. The clients information and requirements will be sent to ASM by the administrator via external notifications (See Fig. 5(b)). ASM will analyze this new information and then decide on the right actions to move from the system's current state to a new one that meets all DM components requirements. This involves either connecting each new client to an existing server that meets the privacy constraint, or else by instantiating a new server on a node within an acceptable location, i.e., dedicated zone.

B. Experimental Setup

The ASM inputs are the infrastructure and the DM system models which are generated based on the use case introduced in Sec.VI-A. This subsection details their attributes.

a) Physical Infrastructure: The global infrastructure is composed of: (i) 20 Edge nodes with CPU cores between 2 and 12 and available RAM between 2 and 24 GB, (ii) 40 less powerful extreme edge nodes close to IoT devices, namely customer premises nodes, characterized by CPU cores

between 1 and 2 and available RAM between 1 and 2 GB, and (iii) up to 1000 links that connect these nodes randomly with bandwidth up to 5/20 Gbps and latency up to 1/5 ms for the physical links that connect the edges nodes and 1/5 Gbps and latency up to 10/20 ms for the physical links that connect edges nodes to customer premises nodes where each node belongs to a specific zone (for the need of privacy scenario).

The nodes zones is given randomly where $DZ1$ to $DZ6$ is reserved of Edge nodes that will host DM servers and $DZ10$ to $DZ20$ is reserved of customer premises nodes.[1]

b) DM Service: For the initial setup we start the DM service with one server and 3 clients managed by this server. After that, the administrator external notifications concerning new IoT clients will arrive following an exponential distribution of mean T_{IA}, where T_{IA} is the mean inter-arrival time of clients arrival notifications (measured in arbitrary time unit). Each DM client has a service time, i.e., the time the client remains in the system is randomly selected following an exponential distribution of mean S. Any notification that cannot be satisfied directly will be kept in a queue for some predefined time, namely Time To Live (**TTL**), waiting for some resources to be released and become available.

In total, our synthetic external notifications workload for the simulations contains request arrivals made of 40/60/80/100 arbitrary clients.The number of clients that arrive or leave is selected randomly between 1 and 4 client at each notification.

The communication between servers and their clients is done via bindings. The processing, RAM, latency and bandwidth requirements for DM components and binding is chosen in random manner with respect to the available infrastructure resources. Moreover, each DM component requires to be started on a specific dedicated zone within the physical infrastructure (more details in Sec. V).

C. Architecture Implementation

We detail here our choices in terms of architecture and language for the development, illustrated in Fig. 6. The software and modules run locally on a workstation with four physical cores and eight logical computing units of the Intel x86-64 architecture, supported by 32GB of DDR4 with 2133MHz RAM with Linux Debian 10 operating system.

For the mapper module, which solves the DM placement problem based on CP, we have chosen Choco as a solver inspired by the works [20] [16], which is a free and open-source Java library dedicated to constraint programming. The placement problem must be modeled in a declarative way by defining the set of constraints to be satisfied in each solution. Then, it will be solved by alternating constraint filtering algorithms with a search mechanism.

The three remaining modules of the ASM, are developed from scratch in Python. The choice of Python is motivated by its wide collection of libraries and its native compatibility with the JSON [30] data format used by the other components of our architecture (messaging bus). The ASM modules communicate with the system administrator via a common RabbitMQ messaging bus [31]. The RabbitMQ bus allows communication via Message Queuing Telemetry Transport (MQTT) which is a messaging protocol widely used in the context of IoT and compliant with Python.

[1]The processing and the memory capacities for each infrastructure node are chosen randomly from set of values inspired by the most common OpenStack and Amazon EC2 instance types and the links bandwidth and latency values also and number of dedicated zones.

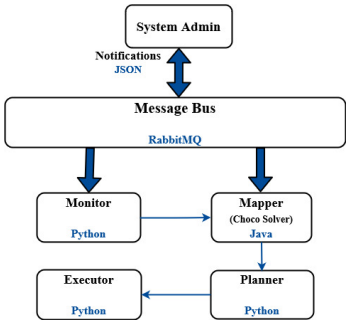


Fig. 6: The Technical Implementation of the Architecture

D. Experimental Results

Our defined simulation scenario aims at initiating a DM service on a physical infrastructure. Later on, at runtime, external notifications about fleet evolution will be launched by the system , triggering our automatic manager to take scaling decisions of the DM servers. All the following experiments were repeated 5 times using 5 different notification workloads with the same parameters.

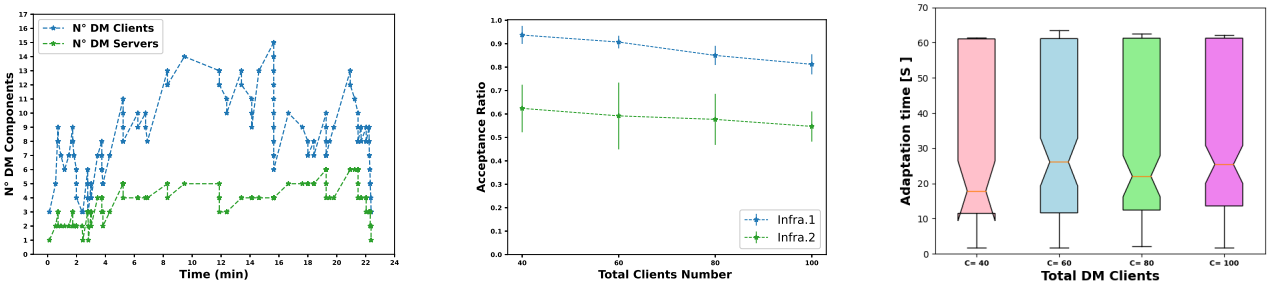
1) ASM Adaptation Characteristic:

We evaluate here how the ASM can adapt the number of DM servers at runtime to fulfill the new DM clients privacy requirements with a total number of 80 clients that want to join the DM system. Figure 7(a) shows the evolution of number of participated servers to adapt to evolution of DM clients through the time. It starts with the initial configuration of 1 server and 3 clients. From this figure we can see an increase in number of servers as new clients join the system (based on their privacy constraint on the acceptable server). However, when new clients arrive and their privacy requirements can be satisfied by the already running servers there is no need to scale out the DM servers. This explains the situations in the figure where we have new incomers and the number of servers does not increase. For example at the time point of 14 to 15 min, the number of servers is 4 while the number of clients in the system goes from 9 to 13 clients.

On the other hand, when the clients number decreased as they leave the system, Fig. 7(a) shows decreasing in the number of servers. This happens when a server has no more clients, e.g., at the time point of 1 min the ASM scale in the number of servers goes from 3 to 2 servers as the the number of clients goes from 9 to 6 clients. However, not all clients departure leads to ASM taking scale in decisions, as their server still have other clients to manage.

2) Acceptance Ratio:

We study the impact of total number of clients that arrived to the system on the ability to satisfy their joining requests



(a) ASM Adaptation Characteristic

(b) Acceptance ratio of DM notifications: Infra.1 & Infra.2 explained in Sec.VI-D2

(c) ASM Adaptation Time

Fig. 7: Experimental Results

and placing them into the Infrastructure. To that aim, we use the acceptance ratio defined as the number of accepted clients arrival notifications over the total number of arrival notifications. Fig. 7(b) shows the acceptance ratio with respect to 4 different total number of clients that ask to join the DM system with two different infrastructures: (i) Infra.1: composed of 20 Edge nodes & 80 customers premises nodes, and (ii) Infra.2 with 20 Edge nodes & 40 customers premises nodes.

In general, and for both infrastructures, we can expect the acceptance ratio to decrease as the total clients number increases since more clients require devoting more physical resources to place the new clients and server when needed. This trend is confirmed by Fig.7(b). However, the decreasing is negligible and this can be explained by the fact of using a waiting queue for the unaccepted notifications to try again when other clients leave the system and some resources become available. Thus, using this queue allows increasing the acceptance ratio and preventing a sudden drop in its value when the number of clients increases from 40 up to 100 clients. However, with more powerful underlying infrastructure (the blue curve in Fig.7(b)) leads to more acceptance ratio as the network can handle more clients before becoming overloaded. It is worth mentioning that even with 40 clients, we do not have an acceptance rate of 1 as the combination of many constraints, namely CPU/RAM/Bandwidth and privacy, prevents the solver from finding a valid solution.

3) ASM Adaptation Time:

To be acceptable, the time spent by ASM to adapt to DM runtime evolution must be at most of the same order of magnitude as the deployment of the VMs that will host the DM components to not impact the deployment time of the DM service. This time includes the generation of a new DM configuration file, taking scale In/Out decisions and finding an acceptable placement of DM components. For that reason, we force the Choco solver in the Mapper to try to find an optimal solution in no more than 1 minute. It is in the same order of magnitude of the typical time to deploy and boot virtual functions in data centers [32].

Figure 7(c) shows the whisker plot of ASM adaptation time to the system administrator external notifications times for 4 different number of DM clients. It shows that the needed time by the ASM to adapt to runtime notifications increases rather linearly with number of DM clients and never exceeds one minute, which is affected by the Mapper module time to find the placement solution. This rather linear increase is because an increase of DM clients number incurs a proportional increase in the total number of DM components. These clients do not come at once, but rather in a random group from 1 to 4 clients together. Thus, the size of the placement problem as the size of the DM components is not impacted.

The spread between median and lower quartile is smaller than the spread between median and upper one as most of placements require more time to find a solution of the new DM configuration or to reject the notification and send it to the waiting queue.

Knowing that the number of arrived clients significantly affects the size of problem, and therefore the needed time by solver to find a solution, this case should be considered as a scalability limitation to our solver. The constraint programming solver could not be fast enough and a new heuristics might be considered to find a near optimal placement.

To best of our knowledge this is the first work on distributed IoT DM at the Edge of the network. Thus, many parameters are set with random values and affect our simulation results. The first parameter is the TTL value: more TTL could increase the acquired acceptance ratio as more resources will be released and become available, but more time is needed to complete a full workload (long waiting queue). The solver maximum allowed time has also a direct impact on both acceptance ratio and ASM adaptation time. Giving more time to Choco solver may increase the acceptance ratio. However, since we are targeting client requests at runtime, it is very important to make a decision as quickly as possible. Moreover, giving more time to the solver cannot guarantee that a solution will be found as the infrastructure might be already overloaded.

Furthermore, since all workloads are randomly generated due to the unavailability of real-world workload- i.e. random arrival rate, service time and resources requirements - running more workloads lead to more consistent results and avoid some ambiguous results, e.g., the median value of the whisker plot for the clients number equal to 60 in Fig. 7(c).

VII. Conclusions and Perspectives

We have addressed the main IoT DM challenges, namely Heterogeneity, dynamicity and scalability. For that purpose, we proposed a self-adaptive DM architecture for IoT with an autonomic manager that is capable of self-scaling the number of DM servers to the fleet changes and requirements at runtime through the distribution of DM operations at the Edge. This autonomic manager relies on a constraint programming solver that is integrated in a feedback loop to decide on DM servers and clients placement while optimizing the infrastructure resources usage. Further, we evaluated this architecture through simulation to the fleet evolution at runtime with a privacy as the target scenario. The results show that our manager is fast enough- only 1 minute- such that one can consider using it in a real environment to handle IoT fleet composition changes at run time and without the need of prior knowledge on the new IoT devices requirements.

For future work, we are investigating our proposal's scaling capability with respect to another scenarios such as physical resources thresholds and DM servers limitations. Also, from the point of view of constraints and models, we want to consider more dynamics (e.g. speed and acceleration of variations) in relation with Control Theory [33]. For interoperability motive, a constraint regarding the management protocol used by the DM server could be considered. Another important enhancement will be testing this architecture with real infrastructure e.g., Grid5000 [34] FIT IoT-LAB Testbed [35].

References

[1] F. Aïssaoui, S. Berlemont, M. Douet, and E. Mezghani, "A semantic model toward smart iot device management," in *Workshops of the International Conf on Advanced Information Networking and Applications*, (Caserta, Italy), pp. 640–650, Springer Publishing, 2020.

[2] T. Perumal, S. K. Datta, and C. Bonnet, "Iot device management framework for smart home scenarios," in *2015 IEEE 4th Global Conf on Consumer Electronics (GCCE)*, (Japan), pp. 54–55, IEEE, 2015.

[3] K. Shea, "Device management in the internet of things–why it matters and how to achieve it." http://www.new-techeurope.com/2017/06/07/device-management-internet-things-matters-achieve/, 2017. Accessed on 2021-02-20.

[4] A. Computing *et al.*, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, no. 2006, pp. 1–6, 2006.

[5] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. USA: Elsevier, 2006.

[6] B. Forum, "Tr-069 cpe wan management protocol." https://www.broadband-forum.org/download/TR-069_Amendment-6.pdf, 2018. Accessed on 2021-02-23.

[7] O. M. Alliance, "Lightweight machine to machine technical specification," *Approved Version*, vol. 1, no. 1, 2017.

[8] M. Elkhodr, S. Shahrestani, and H. Cheung, "The internet of things: new interoperability, management and security challenges," *arXiv preprint arXiv:1604.04824*, vol. abs/1604.04824, p. 85–102, 2016.

[9] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, 2017.

[10] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog orchestration for internet of things services," *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, 2017.

[11] J. H. Ziegeldorf, O. G. Morchon, and K. Wehrle, "Privacy in the internet of things: threats and challenges," *Security and Communication Networks*, vol. 7, no. 12, pp. 2728–2742, 2014.

[12] N. Ayeb, E. Rutten, S. Bolle, T. Coupaye, and M. Douet, "Towards an autonomic and distributed device management for the internet of things," in *IEEE 4th International Workshops on Foundations and Applications of Self* Systems*, (Sweden), pp. 246–248, IEEE, 2019.

[13] N. Ayeb, E. Rutten, S. Bolle, T. Coupaye, and M. Douet, "Coordinated autonomic loops for target identification, load and error-aware device management for the iot," in *15th Conference on Computer Science and Information Systems (FedCSIS)*, (Bulgaria), pp. 491–500, IEEE, 2020.

[14] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[15] M. Tahir, Q. M. Ashraf, and M. Dabbagh, "Towards enabling autonomic computing in iot ecosystem," in *IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress*, (Japan), IEEE, 2019.

[16] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Surveys*, 2020.

[17] B. Donassolo, *IoT Orchestration in the Fog.(L'orchestration des applications IoT dans le Fog).* PhD thesis, Grenoble Alpes University, France, 2020.

[18] S. Challita, F. Paraiso, and P. Merle, "A study of virtual machine placement optimization in data centers," in *7th International Conference on Cloud Computing and Services Science*, (Porto, Portugal), pp. 343–350, INSTICC, 2017.

[19] B. Donassolo, I. Fajjari, A. Legrand, and P. Mertikopoulos, "Load aware provisioning of iot services on fog computing platform," in *ICC IEEE International Conf on Communications*, (China), pp. 1–7, IEEE, 2019.

[20] F. A. Salaht, F. Desprez, A. Lebre, C. Prud'Homme, and M. Abderrahim, "Service placement in fog computing using constraint programming," in *IEEE International Conf on Services Computing*, (Italy), IEEE, 2019.

[21] Y. Xia, *Combining Heuristics for Optimizing and Scaling the Placement of IoT Applications in the Fog*. PhD thesis, Université Grenoble Alpes, 2018.

[22] C. Prud'homme, J.-G. Fages, and X. Lorca, "Choco solver documentation," *TASC, INRIA Rennes, LINA CNRS UMR*, vol. 6241, 2016.

[23] L. Perron and V. Furnon, "Google's or-tools," 2019.

[24] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "Ibm ilog cp optimizer for scheduling," *Constraints*, vol. 23, no. 2, pp. 210–250, 2018.

[25] S. R. Department, "Internet of things- active connections worldwide 2015-2025." https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/, Jan 2021. Accessed on 2021-02-23.

[26] K. Fizza, N. Auluck, A. Azim, M. A. Maruf, and A. Singh, "Faster ota updates in smart vehicles using fog computing," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, (New York, NY, USA), pp. 59–64, Association for Computing Machinery, 2019.

[27] P. Shaw, "A constraint for bin packing," in *International conference on principles and practice of constraint programming*, (Berlin), pp. 648–662, Springer, 2004.

[28] J.-C. Régin, "A filtering algorithm for constraints of difference in csps," in *AAAI*, (USA), pp. 362–367, American Association for AI, 1994.

[29] G. Pesant, "A regular language membership constraint for finite sequences of variables," in *International Conf on principles and practice of constraint programming*, (Heidelberg), pp. 482–495, Springer, 2004.

[30] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of json schema," in *Proceedings of the 25th International Conference on World Wide Web*, (Republic and Canton of Geneva, CHE), pp. 263–273, International World Wide Web Conferences Steering Committee, 2016.

[31] A. RabbitMQ, "Messaging that just works - rabbitmq," 2020. Accessed 07-June-2021.

[32] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*, (Honolulu, HI, USA), pp. 423–430, IEEE, 2012.

[33] M. Litoiu, M. Shaw, G. Tamura, N. M. Villegas, H. Müller, H. Giese, R. Rouvoy, and E. Rutten, "What Can Control Theory Teach Us About Assurances in Self-Adaptive Software Systems?," in *Software Engineering for Self-Adaptive Systems 3: Assurances*, vol. 9640, Springer, May 2017.

[34] D. Balouek, A. C. Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, and L. Nussbaum, "Adding virtualization capabilities to the grid'5000 testbed," in *International Conf on Cloud Computing and Services Science*, pp. 3–20, Springer, 2012.

[35] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "Fit iot-lab: A large scale open experimental iot testbed," in *IEEE 2nd World Forum on IoT*, (Italy), pp. 459–464, IEEE, 2015.