Supporting Application Requirements in Cloud-based IoT Information Processing

Sabrina De Capitani di Vimercati, Giovanni Livraga, Vincenzo Piuri, Pierangela Samarati and Gerson A. Soares
Computer Science Department, Università degli Studi di Milano, Crema, Italy
{*sabrina.decapitani, giovanni.livraga, vincenzo.piuri, pierangela.samarati, gerson.soares*}*@unimi.it*

Keywords: IoT Information Processing, Cloud Computing, Application Requirements, Requirement Dependencies, Service Level Agreement

Abstract: IoT infrastructures can be seen as an interconnected network of sources of data, whose analysis and processing can be beneficial for our society. Since IoT devices are limited in storage and computation capabilities, relying on external cloud providers has recently been identified as a promising solution for storing and managing IoT data. Due to the heterogeneity of IoT data and applicative scenarios, the cloud service delivery should be driven by the requirements of the specific IoT applications. In this paper, we propose a novel approach for supporting application requirements (typically related to security, due to the inevitable concerns arising whenever data are stored and managed at external third parties) in cloud-based IoT data processing. Our solution allows a subject with an authority over an IoT infrastructure to formulate conditions that the provider must satisfy in service provisioning, and computes a SLA based on these conditions while accounting for possible dependencies among them. We also illustrate a CSP-based formulation of the problem of computing a SLA, which can be solved adopting off-the-shelves CSP solvers.

1 INTRODUCTION

Thanks to the achievements recently obtained by the advancements of the ICTs, the Internet of Things (IoT) undoubtedly represents a promising opportunity to build powerful and ubiquitous computing platforms. Leveraging the wide availability of new technologies, such as the RFID and wireless sensor devices, the IoT provides a myriad of common objects with sensing and connectivity capabilities (Jiang et al., 2014). In this regard, devices in the IoT can be seen as a large network of interconnected sources of data, whose processing and analysis can be beneficial for individuals as well as for public and private companies and governmental agencies. For instance, by analyzing the concentration of pollutants sensed by a sensor network, a municipality can constantly keep the quality of air under control in its area (De Capitani di Vimercati et al., 2013), and enforce corrective measures when some pollutants reaches a threshold level. As another example, pervasive healthcare applications use network of body sensors to generate health information about patients that can be remotely monitored to the benefits of the patients themselves (Doukas and Maglogiannis, 2012).

While IoT devices are designed and built to be interconnected, and are able to communicate with the external world, still their storage capacity and computational power are limited. Since the amount of data they generate can be huge (Ma et al., 2012), important challenges related to where and how these data can be stored and processed need to be solved. By making fast and scalable storage and processing infrastructures available at affordable costs (Jhawar and Piuri, 2012), cloud computing has recently been identified as a promising solution for storing and managing IoT information (e.g., (Jiang et al., 2014; Ma et al., 2012; Botta et al., 2014)). The interaction between a Cloud Provider (CP) offering a service and the subjects to which the service is delivered is usually based on Service Level Agreements (SLAs). A SLA represents a contract between the CP and the subjects on different functional/non-functional properties of the provided service. However, relying on pre-defined SLAs might represent a limitation in the context of the IoT, due to the richness and diversity of collected IoT data, and the heterogeneity of the applicative scenarios. For instance, differently from traditional outsourcing scenarios in which oftentimes a bulk data collection is entirely transmitted to the provider at outsourcing time, an IoT application for pollutant monitoring might require timely transmission to the CP of each measurement as soon as it is captured by a sensor, 24/7.

Building on these observations, in this paper we aim at bridging the gap between IoT and cloud computing by proposing an approach for supporting specific application requirements in the definition of a SLA. Our solution allows a subject with an authority over an IoT infrastructure to specify her application requirements to a CP. Typical requirements relate to security, as common whenever data storage and management are delegated to an external (and possibly not fully trusted) CP. The SLA between the subject and the CP, rather than being based on a pre-defined model produced by the CP, can therefore be established by taking into consideration all specific requirements characterizing the application. This problem is however complicated by the fact that the satisfaction of some requirements might depend on the satisfaction of other requirements, of which the requesting subject might be unaware. For instance, to ensure a response time less than a given threshold, due to its hardware and software configurations a CP might not be able to provide other features (e.g., the encryption of the communication channels). Our solution overcomes this problem by taking into consideration dependencies among requirements in the establishment of the SLA. We also propose a formulation of the problem of determining a SLA that is consistent with the given requirements as a Constraint Satisfaction Problem (CSP), allowing for the adoption of existing solvers to compute a solution.

The remainder of this paper is organized as follows. Section 2 presents our problem, a motivating example, and a sketch of the approach. Section 3 illustrates how requirements for the CP and dependencies among requirement conditions can be specified, and formally defines the problem of computing a valid SLA. Section 4 discusses how a valid SLA can be computed and describes a translation of the problem into a CSP. Section 5 discusses related work. Finally, Section 6 concludes the paper.

2 PROBLEM STATEMENT AND SKETCH OF THE APPROACH

In the remainder of this paper, we will refer our examples to a municipality owning a sensor network to measure air pollutants in its area. Each sensor measures specific pollutants at regular time intervals, and the recorded measurements need to be collected and analyzed to set appropriate countermeasures (e.g., restricting vehicles circulation) when needed. Since sensors have limited storage capacity, the municipality aims at relying on an external CP to store and manage the collected data. Outsourced measurements need to be retrieved by the municipality health office whenever needed and, since timely retrieval is a critical factor for fast analysis of air quality, the municipality wishes the CP to ensure a maximum response time to requests. In addition, since the outsourced measurements are considered sensitive information (the existence of correlations between high levels of air pollutants in a certain area and respiratory diseases of citizens living nearby is well known), the municipality wishes that data be either: *i)* physically stored in a chosen trusted country, or *ii)* physically stored at a CP audited for security every week; or *iii)* encrypted by the CP (sensors have limited computational resources to do so[1]). These requirements set the parameters of the service that the CP provides to the municipality and are part of the SLA between the CP and the municipality (hereinafter, the *user* of the service). Figure 1 illustrates our reference scenario.

The SLA establishment starts with the communication to the CP of the application requirements imposing arbitrary conditions on functional/non-funtional properties to be satisfied in the service provision. For instance, in our running example the application requirements of the municipality will include a condition restricting the *response time* to a maximum value. Upon receiving the application requirements, the CP can check whether it can satisfy them and, if this is the case, the conditions in the requirements are inserted into a SLA on which both the CP and the requesting party can agree. If the CP cannot satisfy the given conditions, a SLA cannot be established.

The process of checking whether the application requirements can be fulfilled can be complicated by the possibility that the enforcement of a condition might be possible only provided that other conditions be also enforced. For instance, to ensure a response time lower than a given threshold, a CP might be able to accept only a limited *number of requests* per time unit. This is due to the fact that the response time of a system is not an isolated property: on the contrary, it is linked to other properties by a *dependency* (such as the rate of requests, which have a clear impact on the responsiveness of a system).

We note that dependencies cannot be assumed to

[1]Since measurements cannot be encrypted before storage, we assume for the sake of the example the municipality to choose a CP among those considered trusted for accessing plaintext data, hence confidentiality is required against intruders/unauthorized third parties.

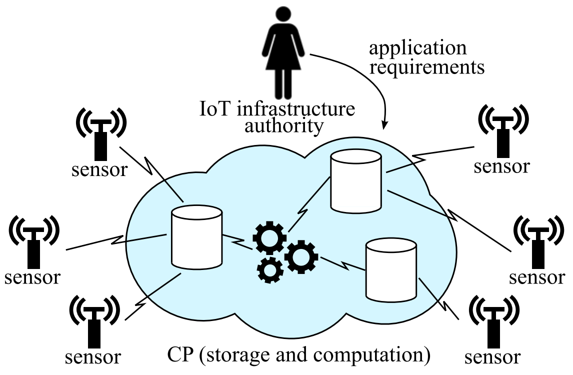


Figure 1: Reference scenario

be known by IoT infrastructure authorities, and taken into account before formulating their application requirements. In fact, they can be provider-dependent, meaning that some dependencies might hold for a given CP while not holding for other ones. While dependencies must therefore be transparent for the users, each CP knows the specific dependencies that hold for its services. To build a valid SLA starting from application requirements, the CP must then check such requirements against possible dependencies.

It is easy to see that the consideration of both application requirements and dependencies in the establishment of a SLA can result in different outcomes: *i)* the requirement conditions can be satisfied as they are (i.e., no dependency is involved) and can be put into a SLA; *ii)* the conditions cannot be satisfied (e.g., the CP does not have resources to fulfil them), and a valid SLA cannot be created; and *iii)* some conditions involve dependencies that require the enforcement of further conditions, which then also need to be inserted into a valid SLA. In the following sections, we will formally model application requirements and dependencies, and then illustrate our approach to compute a valid SLA.

3 PROBLEM MODEL

Application requirements are formulated as Boolean formulas over conditions defined on attributes that represent (functional or non-functional) properties characterizing the cloud services and are taken from a common/shared ontology (Galster and Bucherer, 2008). With reference to our running example, `srv_loc`, `resp_time`, `encr`, `sec_audit`, and `access_log` are examples of attributes (modeling respectively the physical location of a server, the response time of the service, the encryption algorithm adopted by the provider, the auditing frequency, and whether accesses are logged) that the municipality can use to define conditions on the required service. Let A be the set of attributes. Each attribute $a \in A$ takes values from its domain $dom(a)$. For instance, $dom(\texttt{srv\_loc})=\{\text{USA, EU}\}$, $dom(\texttt{resp\_time})=\{10, 20, 50\}$, $dom(\texttt{encr})=\{\text{AES, DES, no}\}$, $dom(\texttt{security\_audit})=\{\text{weekly, monthly, no}\}$, $dom(\texttt{access\_log})=\{\text{yes, no}\}$. A condition defined over an attribute restricts the values that the property represented by the attribute can assume in the provision of the service. A condition is formally defined as follows.

Definition 3.1 (Condition). *Given a set A of attributes, an attribute $a \in A$ with domain $dom(a)$, and a value $val \in dom(a)$, a* condition *c over a is a term of the form $c : \langle a \text{ op } val\rangle$, with $\text{op} \in \{=, \neq, <, \leq, >, \geq\}$ a comparison operator.*

Figure 2(a) illustrates a set of conditions for our running example. For instance, c_5:$\langle\texttt{resp\_time}<10\rangle$ and c_{10}:$\langle\texttt{srv\_loc}=\text{USA}\rangle$ model two conditions demanding that the service exhibits a response time lower than 10 milliseconds (c_5) and uses a storage server being located in USA (c_{10}).

Application requirements can be composed of different conditions over different attributes. More precisely, by interpreting each condition c as a Boolean variable, an application requirement can be naturally expressed as a Boolean formula over such variables. For simplicity but without loss of generality, we assume requirements to be in disjunctive normal form (DNF). An application requirement is formally defined as follows.

Definition 3.2 (Application requirement). *Given a set $C = \{c_1, \ldots, c_n\}$ of conditions over a set A of attributes, an* application requirement *$\mathcal{R}$ over C is a formula of the form $\bigvee_{i=1}^{m}(\bigwedge_{j=1}^{k(i)} c_{i_j})$, with $k(i)$ the number of conditions of the i^{th} clause, and $c_{i_j} \in C$.*

For instance, considering the conditions in Figure 2(a), the application requirement of our running example can be formulated as $\mathcal{R}$: $(c_5 \wedge c_{10}) \vee (c_5 \wedge c_2) \vee (c_5 \wedge c_6)$. Intuitively, $\mathcal{R}$ states that to satisfy the requirements of the municipality, the cloud service should exhibit a response time lower than 10 milliseconds (c_5) *and* use a storage server located in USA (c_{10}), *or* exhibit a response time lower than 10 milliseconds (c_5) *and* use AES for encryption (c_2), *or* exhibit a response time lower than 10 milliseconds (c_5) *and* weekly execute security auditing (c_6).

As already mentioned in Section 2, a SLA should also consider possible dependencies related to the

(a)

c_1: $\langle$proc_num$\geq$2$\rangle$	c_5: $\langle$resp_time$<$10$\rangle$	c_9: $\langle$backup=daily$\rangle$
c_2: $\langle$encr=AES$\rangle$	c_6: $\langle$sec_audit=weekly$\rangle$	c_{10}:$\langle$srv_loc=USA$\rangle$
c_3: $\langle$encr=no$\rangle$	c_7: $\langle$access_log=yes$\rangle$	c_{11}:$\langle$storage$<$100TB$\rangle$
c_4: $\langle$req_rate$<$1/min$\rangle$	c_8: $\langle$backup=no$\rangle$	c_{12}:$\langle$storage$\geq$100TB$\rangle$

(b)

d_1: $\langle$srv_loc=USA$\rangle\rightsquigarrow\langle$storage$<$100TB$\rangle$
d_2: $\langle$resp_time$<$10$\rangle\rightsquigarrow\langle$backup=no$\rangle$ $\wedge$ $\langle$req_rate$<$1/min$\rangle$ $\wedge$ $\langle$encr=no$\rangle$
d_3: $\langle$encr=AES$\rangle\rightsquigarrow\langle$proc_num$\geq$2$\rangle$
d_4: $\langle$backup=daily$\rangle\rightsquigarrow\langle$storage$\geq$100TB$\rangle$
d_5: $\langle$security_audit=weekly$\rangle\rightsquigarrow\langle$access_log=yes$\rangle$

Figure 2: Conditions for our running example (a) and dependencies over them (b)

conditions included in $\mathcal{R}$. Dependencies capture generic relationships among attributes, implying that the enforcement of a condition over an attribute *depends on* the enforcement of another condition over another attribute. For instance, with reference to our running example, attributes resp_time and req_rate (modeling, respectively, the response time of the system and the rate of requests) are linked by a dependency. If $\mathcal{R}$ includes a condition over resp_time, then a valid SLA should also include a condition over req_rate.

While attribute dependencies can be considered to always hold (e.g., the responsiveness of a service is always impacted by rate of requests it receives), the specific conditions holding for the attributes involved in a dependency can vary depending on the CP (e.g., a CP with a set of servers running in parallel might accept more requests per time unit than another CP with a single server). Upon receiving an application requirement $\mathcal{R}$ from the IoT infrastructure authority, the CP must verify whether the conditions in $\mathcal{R}$ imply other conditions due to the presence of dependencies. Note that dependencies can model both *incompatibilities* among conditions (i.e., enforcing a condition over a_i does not allow to enforce another condition over a_j) and *implications* among them (i.e., enforcing a condition over a_i requires the enforcement of another condition over a_j). Building on our interpretation of conditions as Boolean variables, a *condition dependency*, meaning an attribute dependency instantiated with conditions over its attributes, is formally defined as follows.

Definition 3.3 (Condition dependency)**.** *Given a set $C = \{c_1,\ldots.c_n\}$ of conditions over a set A of attributes, a* condition dependency *d over C is defined as d : $c_h \rightsquigarrow (\bigvee_{i=1}^{m}(\bigwedge_{j=1}^{k(i)} c_{i_j}))$, with $k(i)$ the number of conditions of the i^{th} clause, and $c_h, c_{i_j} \in C$.*

A dependency $c_h \rightsquigarrow (\bigvee_{i=1}^{m}(\bigwedge_{j=1}^{k(i)} c_{i_j}))$ can be interpreted as a material implication: if condition c_h is satisfied, then also $\bigvee_{i=1}^{m}(\bigwedge_{j=1}^{k(i)} c_{i_j})$ must be satisfied. Figure 2(b) illustrates five condition dependencies for our running example defined over the set of conditions in Figure 2(a). Dependency d_1 states that providing a server located in USA implies a maximum storage capacity of 100TB. Dependency d_2 states that a response time lower than 10ms is incompatible with the execution of backups and of encryption operations (incompatibilities), and imposes a maximum rate of requests of 1 per minute. Dependency d_3 states that to provide AES encryption, the server must have at least two processors. Dependency d_4 states that a daily backup requires a storage capacity greater than or equal to 100TB. Dependency d_5 states that to ensure a weekly auditing process, accesses should be logged. Conditions in a dependency can involve attributes under the control of either the CP (e.g., $\langle$storage$<$100TB$\rangle$ in d_1) or the IoT infrastructure authority/users (e.g., $\langle$req_rate$<$1/min$\rangle$ in d_2, being the request rate dependent on the operations of the authority/users). For the sake of readability, in the remainder of this paper, we will denote DNF formulas over a set $\{c_i,\ldots,c_j\}$ of conditions with notation $\mathcal{P}(c_i,\ldots,c_j)$.

In our scenario, given a set $C = \{c_1,\ldots,c_n\}$ of conditions, a SLA is naturally represented as a set $\{c_1,\ldots,c_k\} \subseteq C$ of conditions, whose enforcement must be guaranteed in the service provision. Note that a SLA should include at most one condition over each attribute $a \in A$ (as otherwise they would be in conflict). We refer to a set of conditions satisfying this property as *well-formed*, as follows.

Definition 3.4 (Well-formed set of conditions)**.** *Given a set C of conditions over a set A of attributes, C is said to be* well-formed *iff $\forall c \in C, \forall a \in A$, $|C_a| \leq 1$, with $C_a \subseteq C$ the conditions over attribute a.*

For instance, with reference to the conditions in Figure 2(a), the set $\{c_2, c_{11}, c_{12}\}$ is not well-formed as c_{11} and c_{12} are defined over the same attribute storage.

Given a set $C = \{c_1,\ldots,c_n\}$ of conditions, an application requirement $\mathcal{R}$ over C, and a set $D =$

$\{d_1,\ldots,d_l\}$ of dependencies over C, our goal is to find a subset of conditions in C that forms a *valid SLA*, meaning that the SLA is well-formed and satisfies both $\mathcal{R}$ and D. Following our logical modeling where conditions are interpreted as Boolean variables, application requirements as Boolean formulas, and dependencies as material implications, we introduce an assignment function $f : C \rightarrow \{0,1\}$ assigning to each condition $c{\in}C$ a value from the set $\{0,1\}$. With a slight abuse of notation, we use f to denote also the list of values assigned by f to the conditions in C. Therefore, given a requirement $\mathcal{R}$ over C, $f(\mathcal{R})$ will denote the result of the evaluation of $\mathcal{R}$ with respect to the values in f. Since $\mathcal{R}$ must be satisfied when assigning values to C to compute a SLA, f is a *correct assignment w.r.t. a requirement* $\mathcal{R}$ iff $f(\mathcal{R}) = 1$. Similarly, a dependency $c_h{\rightsquigarrow}\mathcal{P}(c_i,\ldots,c_j)$ is satisfied provided that, if $f(c_h) = 1$, then $f(\mathcal{P}(c_i,\ldots,c_j)) = 1$. Therefore, f is a *correct assignment w.r.t. a set D of dependencies* iff $f(d) = 1, \forall d \in D$.

A SLA is then interpreted as a complete value assignment over the conditions in C, where the conditions included in the SLA are those assigned value 1 by f. Our problem of determining a valid SLA (vSLA) given an application requirement $\mathcal{R}$ and a set D of dependencies over a set C of conditions can therefore be interpreted as finding a value assignment f being correct w.r.t. $\mathcal{R}$ and D, and such that the set of conditions assigned value 1 by f be well-formed, as formally defined as follows.

Problem 3.1 (vSLA). *Given a set $C = \{c_1,\ldots,c_n\}$ of conditions, an application requirement $\mathcal{R}$ over C, and a set $D = \{d_1,\ldots,d_l\}$ of dependencies over C, determine (if it exists) a value assignment f to the conditions in C s.t.:*

- $f(\mathcal{R}) = 1$ *(*requirement satisfaction*);*
- $f(d) = 1, \forall d \in D$ *(*dependency satisfaction*);*
- $\{c_i \in C : f(c_i) = 1\}$ *is well-formed according to Definition 3.4 (*conflict satisfaction*).*

With reference to our running example and the dependencies in Figure 2(b), an example of an assignment f solving the vSLA problem assigns value 1 to conditions c_3, c_4, c_5, c_8, c_{10}, c_{11}, and value 0 to conditions c_1, c_2, c_6, c_7, c_9, and c_{12}. Such an assignment corresponds to a SLA including the following conditions (see Figure 2(a)): ⟨`encr`=no⟩, ⟨`req_rate`<1/min⟩, ⟨`resp_time`<10⟩, ⟨`backup`=no⟩, ⟨`srv_loc`=USA⟩, ⟨`storage`<100TB⟩. It is easy to see that such a SLA enforces the requirement of the municipality and the conditions requested by the dependencies, and that it is well-formed. In the next section, we illustrate our approach to compute a valid SLA taking dependencies into account.

4 COMPUTING A VALID SLA

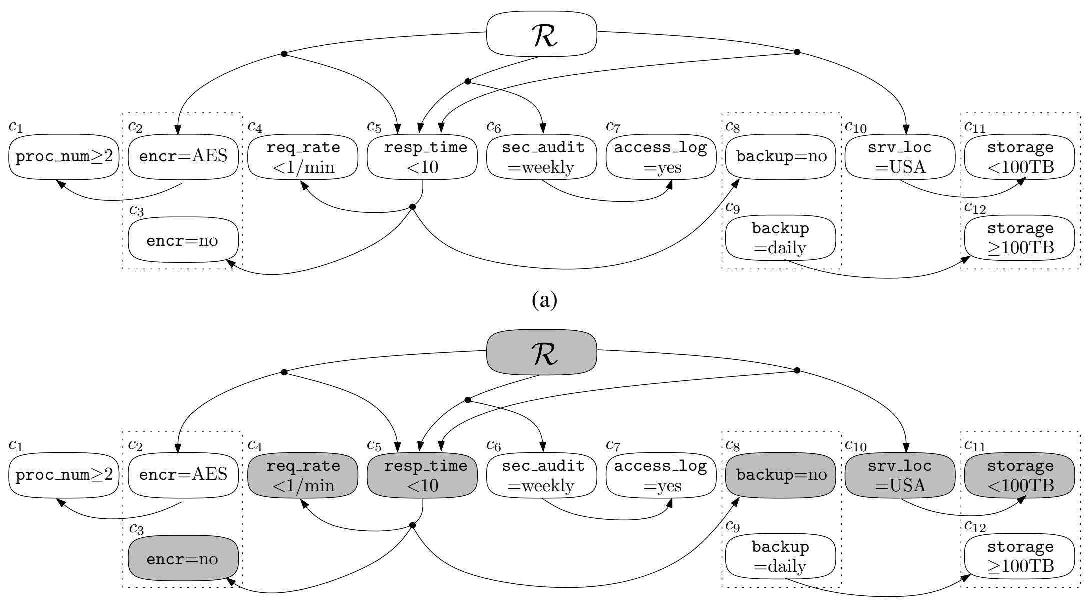
To reason about the conditions of Problem 3.1, we first give an intuitive graphical representation. We then present a solution based on a translation of the problem as a CSP.

4.1 Graph Modeling

Our problem can be naturally represented through a mixed and colored hypergraph representing the input of Problem 3.1. Such mixed hypergraph $G(V,E,E^u)$, with V the set of vertices and E (E^u, resp.) the set of directed (undirected, resp.) hyperedges, is defined as follows.

- Each condition appearing in $\mathcal{R}$ and in the set $D = \{d_1,\ldots,d_l\}$ of dependencies holding for the CP, as well as the requirement $\mathcal{R}$, correspond to a vertex $v \in V$;
- each dependency $d : c_h \rightsquigarrow \mathcal{P}(c_i,\ldots,c_j)$ where $\mathcal{P}(c_i,\ldots,c_j)$ is composed of m OR-ed terms is translated into m directed hyperedges in E where the i^{th} hyperedge connects c_h to all conditions of the i^{th} term, $i = \{1,\ldots,m\}$;
- the requirement $\mathcal{R}$, composed of m OR-ed terms, is translated into m directed hyperedges in E where the i^{th} hyperedge connects vertex $\mathcal{R}$ to all conditions of the i^{th} term, $i = \{1,\ldots,m\}$;
- for each attribute a appearing in the conditions in the graph, the set C_a of conditions defined over the same attribute a is translated in an undirected hyperedge in E^u connecting all conditions in C_a.

Figure 3(a) illustrates the hypergraph modeling our running example, where hyperedges in E (E^u, resp.) are represented as arrows (dotted boxes, resp.) linking (surrounding, resp.) the involved conditions. The computation of a solution to Problem 3.1 can be interpreted as a coloring of the vertices of the hypergraph, starting from the vertex representing $\mathcal{R}$ and recursively propagating the color through the directed hyperedges in E. Note that, when more than one hyperedge originates from the same vertex $v{\in}V$, it is sufficient to propagate the color through one hyperedge (recall that m hyperedges correspond to m OR-ed terms). Such color propagation through directed hyperedges guarantees that the colored vertices represent a set of conditions satisfying the first two conditions in Problem 3.1. In fact, directed hyperedges



(b)

Figure 3: Graphical representation of Problem 3.1 for our running example

link all conditions included in the OR-ed terms in $\mathcal{R}$, and also conditions in the OR-ed terms in the dependencies enabled by the coloring of a term in $\mathcal{R}$. Figure 3(b) illustrates the hypergraph of Figure 3(a) after the color propagation from $\mathcal{R}$ through the hyperedge representing $(c_5 \wedge c_{10})$.

Once the color has propagated through the directed hyperedges in E, undirected hyperedges in E^u can be exploited to check the satisfaction of Condition 3 in Problem 3.1 (conflict satisfaction). By restricting $G(V,E,E^u)$ to $G'(V,E^u)$, Condition 3 is satisfied iff the vertices colored in the previous step through the edges in E represent an *independent set* for G'. In fact, since all conditions defined over the same attribute are connected through an undirected hyperedge, if for every hyperedge $[v_i,\dots,v_j]$ in E^u, *at most* one vertex $v_x \in \{v_i,\dots,v_j\}$ is colored, then the set of colored vertices in G' includes at most one condition for every attribute. Figure 3(b) shows that the set of colored vertices form an independent set for $G'(V,E^u)$. In the remainder of this section, we illustrate our approach to compute a solution to Problem 3.1.

4.2 CSP Formulation

To find a solution to our problem, we represent it as a Constraint Satisfaction Problem (CSP), which can then be conveniently solved with off-the-shelf CSP solvers (De Capitani di Vimercati et al., 2014). The CSP is formulated as follows: given a triple $\langle X,Z,K\rangle$, with X a set of variables, Z the domain of variables in X, and K a set of constraints over X, find an assignment $w : X \to Z$ that satisfies all the constraints in K. Our translation interprets:

- all conditions appearing in $\mathcal{R}$ and in the set D of dependencies as the set X of variables;
- the set of integers $\{1,0\}$ as the domain Z of the variables in X;
- the requirement $\mathcal{R}$, the set D of dependencies, and the conflicts among conditions as the set K of constraints.

A solution to the problem so defined corresponds to a value assignment $w(c)$ to all conditions in C such that w satisfies all the constraints in K. With reference to our hypergraph, X corresponds to the set V of vertices excluding $\mathcal{R}$, Z corresponds to the domain of colors (1 translates to gray), K corresponds to $\mathcal{R}$ and the dependencies and conflicts modeled through hyperedges, and w corresponds to the coloring function.

We now illustrate how application requirements, dependencies, and conflicts can be translated into equivalent CSP constraints.

Requirements. Given an application requirement $\mathcal{R}$:$\bigvee_{i=1}^{m}(\bigwedge_{j=1}^{k(i)} c_{i_j})$ composed of m OR-ed terms, then *all* conditions in *at least* one of these terms must be included in a valid SLA. In terms of the CSP assignment function w, at least one of the m terms must be assigned value 1. Formally, a requirement $\mathcal{R}$ is interpreted as

	Input	CSP formulation
Requirement	$(c_5 \wedge c_{10}) \vee (c_5 \wedge c_2) \vee (c_5 \wedge c_6)$	$(c_5 = c_{10} = 1) \vee (c_5 = c_2 = 1) \vee (c_5 = c_6 = 1)$
Dependencies	$c_5 \rightsquigarrow c_8 \wedge c_4 \wedge c_3$	$(c_5 = 0) \vee (c_8 = c_4 = c_3 = 1)$
	$c_{10} \rightsquigarrow c_{11}$	$(c_{10} = 0) \vee (c_{11} = 1)$
	$c_2 \rightsquigarrow c_1$	$(c_2 = 0) \vee (c_1 = 1)$
	$c_9 \rightsquigarrow c_{12}$	$(c_9 = 0) \vee (c_{12} = 1)$
	$c_6 \rightsquigarrow c_7$	$(c_6 = 0) \vee (c_7 = 1)$
Conflicts	$C_{\texttt{storage}} = \{c_{11}, c_{12}\}$	$(c_{11} = 0 \wedge c_{12} = 1) \vee c_{12} = 0$
	$C_{\texttt{backup}} = \{c_9, c_8\}$	$(c_8 = 0 \wedge c_9 = 1) \vee c_9 = 0$
	$C_{\texttt{encr}} = \{c_2, c_3\}$	$(c_3 = 0 \wedge c_2 = 1) \vee c_2 = 0$

Figure 4: Requirement, dependencies, and conflicts together with their CSP formulation for our running example

$$\bigvee_{i=1}^{m} (c_{i_1} = \ldots = c_{i_{k(i)}} = 1)$$

Dependencies. Given a dependency $d{:}c_h \rightsquigarrow \mathcal{P}(c_i, \ldots, c_j)$, with $\mathcal{P}(c_i, \ldots, c_j) = \bigvee_{i=1}^{m} (\bigwedge_{j=1}^{k(i)} c_{i_j})$, then *all* conditions in *at least* one of the m OR-ed terms must be included in a valid SLA if c_h is also included. In terms of the CSP assignment function w, at least one of the m terms must be assigned value 1, if also c_h is assigned value 1. Formally, a dependency $c_h \rightsquigarrow \mathcal{P}(c_i, \ldots, c_j)$ is interpreted as

$$(c_h = 0) \vee \left(\bigvee_{i=1}^{m} (c_{i_1} = \ldots = c_{i_{k(i)}} = 1) \right)$$

Conflicts. Given the set C_a of conditions over attribute a, then at most one condition $c \in C_a$ can be included in a valid SLA. In terms of the CSP assignment function w, at most one condition $c \in C_a$ must be assigned value 1. Formally, a set $C_a = \{c_1, \ldots, c_k\}$ is interpreted as

$$(c_1 = \cdots = c_k = 0) \vee$$
$$\left(\bigvee_{i=1}^{k} (c_i = 1 \wedge (c_{j_1} = \cdots = c_{j_{k-1}} = 0)) \right),$$
$$c_{j_l} \in \{c_1, \ldots, c_k\} \setminus \{c_i\}$$

Note that CSP constraints correspond to the conditions of Problem 3.1, and a function w satisfying them corresponds to a correct value assignment f of Problem 3.1. Figure 4 illustrates the CSP constraints for our running example. A valid SLA will include all conditions assigned value 1 by function w.

The CSP translation illustrated above can be solved by adopting any CSP solver. We formulated the constraints in Figure 4 adopting the well-known SWI-Prolog interpreter, enriched with the `CLP(fd)` Prolog library (Triska, 2012), and obtained a result assigning value 1 to the colored vertices in Figure 3(b), corresponding to a valid SLA (i.e., a solution to Problem 3.1).

5 RELATED WORK

The combination of IoT and cloud computing raises a number of issues, as recently investigated by the research community (e.g., (Biswas and Giaffreda, 2014)), which has proposed several solutions for bridging the gap between these two technologies. These solutions focus on a variety of problems, including cloud-based storage of IoT data (e.g., focusing on storage efficiency and supporting structured and unstructured data (Jiang et al., 2014)), efficient querying of IoT data stored in the cloud (e.g., proposing an index-based framework also supporting simultaneous multi-dimensional queries (Ma et al., 2012)), IoT service delivery on the cloud (e.g., proposing a PaaS framework focusing on efficient service delivery and multi-tenancy (Li et al., 2013)), efficient protocols linking IoT and cloud (e.g., proposing a CoAP-based system architecture for scalable cloud services in the IoT (Kovatsch et al., 2014)). While related, our work addresses a different problem focused on supporting application requirements in cloud processing of IoT information. To this end, all these techniques can be seen as complementary to ours, which can be used to establish a SLA before adopting them.

Another line of research connected to our work is related to the problem of establishing/assessing SLA in cloud computing (e.g., (Foresti et al., 2015)). The work in (Garg et al., 2013) aims at assessing cloud services based on customer requirements. While considering relationships between (sub-)attributes, which may resemble our dependencies, the problem addressed is different as it aims at prioritizing cloud services. Dependencies have been considered also in (Kotsokalis et al., 2010), where however they model different relationships (e.g., failures at the infrastructure and software levels). The work in (Li et al., 2010) illustrates a framework for comparing different cloud providers over different performance indicators, focusing specifically on the performance and cost of the providers. The proposal in (Jhawar et al., 2012), while sharing with our work the support for user-based constraints in cloud computing (which may recall our application requirements), focuses on a different problem related to cloud resource management where constraints model security requirements.

6 CONCLUSIONS

In this paper, we have proposed an approach for supporting application requirements in cloud-based IoT information processing. Our solution allows an IoT infrastructure authority to specify arbitrary requirements that must be satisfied by the CP during the service provision and computes a SLA based on them. Our approach takes into account possible dependencies among requirements that might require the satisfaction of further conditions. We have also provided a CSP-based approach for solving our problem.

ACKNOWLEDGEMENTS

This work was supported in part by: the EC within 7FP (grant 312797 "ABC4EU"), and H2020 (grant 644597 "ESCUDO-CLOUD"), and by the Italian MIUR within PRIN "GenData 2020" (2010RTFWBH), and the Brazilian CAPES (grant BX5949/13-0).

REFERENCES

Biswas, A. and Giaffreda, R. (2014). IoT and cloud convergence: Opportunities and challenges. In *Proc. of the 2014 IEEE World Forum on Internet of Things (IEEE WF-IoT 2014)*, Seoul, South Korea.

Botta, A., de Donato, W., Persico, V., and Pescapé, A. (2014). On the integration of cloud computing and Internet of Things. In *Proc. of the 2nd International Conference on Future Internet of Things and Cloud (FiCloud 2014)*, Barcelona, Spain.

De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., and Samarati, P. (2014). Fragmentation in presence of data dependencies. *IEEE Transactions on Dependable and Secure Computing (IEEE TDSC)*, 11(6):510–523.

De Capitani di Vimercati, S., Genovese, A., Livraga, G., Piuri, V., and Scotti, F. (2013). Privacy and security in environmental monitoring systems: Issues and solutions. In Vacca, J., editor, *Computer and Information Security Handbook, 2nd Edition*. Morgan Kaufmann.

Doukas, C. and Maglogiannis, I. (2012). Bringing IoT and cloud computing towards pervasive healthcare. In *Proc. of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012)*, Palermo, Italy.

Foresti, S., Piuri, V., and Soares, G. (2015). On the use of fuzzy logic in dependable cloud management. In *Proc. of the 2015 IEEE Conference on Communications and Network Security (IEEE CNS 2015)*, Florence, Italy. poster.

Galster, M. and Bucherer, E. (2008). A taxonomy for identifying and specifying non-functional requirements in service-oriented development. In *Proc. of the 2008 IEEE Congress on Services (IEEE SERVICES 2008)*, Hawaii, USA.

Garg, S. K., Versteeg, S., and Buyya, R. (2013). A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 29(4):1012–1023.

Jhawar, R. and Piuri, V. (2012). Fault tolerance management in iaas clouds. In *Proc. of the 1st IEEE-AESS Conference in Europe about Space and Satellite Telecommunications (ESTEL 2012)*, Rome, Italy.

Jhawar, R., Piuri, V., and Samarati, P. (2012). Supporting security requirements for resource management in cloud computing. In *Proc. of the 15th IEEE International Conference on Computational Science and Engineering (IEEE CSE 2012)*, Paphos, Cyprus.

Jiang, L., Xu, L. D., Cai, H., Jiang, Z., Bu, F., and Xu, B. (2014). An IoT-oriented data storage framework in cloud computing platform. *IEEE Transactions on Industrial Informatics (IEEE TII)*, 10(2):1443–1451.

Kotsokalis, C., Yahyapour, R., and Gonzalez, M. (2010). SAMI: The SLA management instance. In *Proc. of the 5th International Conference on Internet and Web Applications and Services (ICIW 2010)*, Barcelona, Spain.

Kovatsch, M., Lanter, M., and Shelby, Z. (2014). Californium: Scalable cloud services for the Internet of Things with CoAP. In *Proc. of the 2014 IEEE International Conference on the Internet of Things (IEEE IOT 2014)*, Cambridge, MA, USA.

Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). Cloudcmp: Comparing public cloud providers. In *Proc. of ACM SIGCOMM 2010*, Melbourne, Australia.

Li, F., Voegler, M., Claessens, M., and Dustdar, S. (2013). Efficient and scalable IoT service delivery on cloud. In *Proc. of IEEE CLOUD 2013*, Santa Clara, CA, USA.

Ma, Y., Rao, J., Hu, W., Meng, X., Han, X., Zhang, Y., Chai, Y., and Liu, C. (2012). An efficient index for massive IoT data in cloud environment. In *Proc. of the 21st ACM International Conference on Information and Knowledge Management (ACM CIKM 2012)*, Maui, Hawaii, USA.

Triska, M. (2012). The finite domain constraint solver of SWI-Prolog. In *Poc. of the 11th International Symposium on Functional and Logic Programming (FLOPS 2012)*, Kobe, Japan.