Application Requirements with Preferences in Cloud-based Information Processing

Sabrina De Capitani di Vimercati, Giovanni Livraga, Vincenzo Piuri
Department of Computer Science
Università degli Studi di Milano – Crema, IT
Email: {sabrina.decapitani, giovanni.livraga, vincenzo.piuri}@unimi.it

***Abstract*—The use of cloud services is typically regulated by a Service Level Agreement (SLA) that defines the specific parameters of the service that will be provided by a Cloud Provider. The ability of users to negotiate a SLA that meets their requirements is a key issue in cloud computing. In this paper, we illustrate an approach for allowing users to specify their requirements and preferences over these requirements, and to consider them in the definition of a SLA.**

I. Introduction

With the progress of cloud technology, users and companies can take full advantage of the variety of scalable and elastic computing and storage services available in the cloud market. The provisioning of a cloud service is usually regulated by a *Service Level Agreement* (SLA) that represents a contract between a Cloud Provider (CP) and the data/service owner, specifying the quality of service and the properties that the Cloud Provider should guarantee. While the definition of a SLA typically relies on predefined templates, in some scenarios there might be the need to provide more flexibility and allowing the derivations of SLAs from the requirements of the specific services to be supported. The work in [1] introduced the idea of supporting SLAs computed starting from application requirements specified by users. The approach proposed also considers dependencies among requirements that cloud providers might imposed (e.g., if data are to be encrypted and processing performed on encrypted data, the response time might be not lower than a given threshold) and proposes a formulation of the problem of computing a valid SLA (i.e., a SLA that satisfies the user-defined requirements and all dependencies possibly enabled by them) as a Constraint Satisfaction Problem.

In the definition of requirements, users can include different combinations of characteristics. For instance, consider a scenario where a manager of a sensor network that measures noise levels in a certain geographical area wishes to use a cloud service offered by a Cloud Provider for storing and managing the collected measurements. To be compliant with laws and regulations, the manager requires that measurements be either stored in the country where they have been collected (e.g., in Italy), or in a different country if stored in encrypted form. While the approach in [1] considers the different alternatives in the requirements to be all equally acceptable, it might be desirable to allow users to specify possible preferences among alternatives, which can be taken into consideration in the definition of the SLA. For instance, the manager might prefer to store the measurements in the country where they are collected rather than abroad. If both options are applicable, the SLA should include the one preferred by the user.

In this paper, we propose an approach for including preferences among alternatives in the specification of application requirements, and illustrate how such preferences can be considered in the computation of a SLA. We also introduce a solution for automatically deriving such preferences based on preferences defined over the values that can be assumed by the service properties. Our solution builds on the modeling and techniques proposed in [1], thus nicely extending the original approach. The remainder of this paper is organized as follows. Section II briefly illustrates the basic concepts related to the modeling of application requirements and dependencies among service properties and provides a brief overview of how computing a SLA satisfying them [1]. Section III describes how preferences among alternative options in an application requirement can be formulated and considered in the computation of a SLA. Section IV illustrates how the preferences illustrated in Section III can be automatically derived from preferences over the values that the service properties can assume. Section V discusses related work. Finally, Section VI concludes the paper.

II. Supporting application requirements in SLAs

We refer our examples to a manager of a sensor network wishing to find a Cloud Provider to store and manage measurements collected through the sensor network. The manager requires that measurements be: *i)* stored in Italy; or *ii)* stored in France and encrypted with AES; or *iii)* stored in UK, encrypted with 3DES, and audited for security every week. These three alternatives form the application requirement of the manager and, according to the proposal in [1], the provisioning of a service offered by a Cloud Provider should be regulated by a SLA that guarantees the satisfaction of at least one of these three options. We represent an *application requirement* as a DNF Boolean formulas over *conditions* defined on *attributes* taken from a common/shared ontology [2] representing properties of the cloud services. Each attribute a in a set A of attributes takes values from a domain $D(a)$. A condition c of the form $c : \langle a \ op \ val \rangle$, with $op \in \{=, \neq, <, \leq, >, \geq\}$, restricts the values that the property represented by a can assume in the provisioning of

A= {loc, encr, audit, max_storage}

D(loc) = {IT, FR, DE, UK}
D(encr) = {AES, 3DES}
D(audit) = {1week, 2weeks, 4weeks}
D(max_storage) = {50TB, 100TB}

(a)

c_1: ⟨loc=IT⟩	c_5: ⟨encr=AES⟩
c_2: ⟨loc=FR⟩	c_6: ⟨encr=3DES⟩
c_3: ⟨loc=DE⟩	c_7: ⟨audit=1 week⟩
c_4: ⟨loc=UK⟩	c_8: ⟨max_storage=100TB⟩

(b)

Fig. 1. An example of a set A of attributes with their domains (a), and a set of conditions over them (b)

the service. Figure 1 illustrates a sample set A of attributes with their domains, and of conditions over the attributes in A. By interpreting each condition c as a Boolean variable, an application requirement $\mathcal{R}$ over a set $C = \{c_1, \ldots, c_n\}$ of conditions can be naturally expressed as a Boolean formula over such variables, where each conjunctive clause represents an alternative. For instance, considering the conditions in Figure 1(b), the application requirement $\mathcal{R}$ of our running example can be formulated as $(c_1) \vee (c_2 \wedge c_5) \vee (c_4 \wedge c_6 \wedge c_7)$.

Given a set $C = \{c_1, \ldots, c_n\}$ of conditions, a SLA should include a subset $\{c_1, \ldots, c_k\}$ of C such that $\mathcal{R}$ is satisfied (i.e., all conditions in at least one of the clauses in $\mathcal{R}$ are included in $\{c_1, \ldots, c_k\}$). The presence of some conditions $\{c_1, \ldots, c_k\}$ in a SLA may possible require also the inclusion of additional conditions due to the existence of dependencies triggered by $\{c_1, \ldots, c_k\}$. Building on the interpretation of conditions as Boolean variables, a *dependency* d over a set $C = \{c_1, \ldots, c_n\}$ of conditions is defined as $d : c_h \rightsquigarrow (\bigvee_{i=1}^{m} (\bigwedge_j c_{i_j}))$, with c_{i_j} the j-th condition of the i^{th} clause, and $c_h, c_{i_j} \in C$. As an example, consider dependencies $d_1 : \langle \texttt{loc} = \text{FR} \rangle \rightsquigarrow \langle \texttt{max\_storage} = \text{100TB} \rangle$, and $d_2 : \langle \texttt{loc} = \text{IT} \rangle \rightsquigarrow \langle \texttt{encr} = \text{AES} \rangle$.

A dependency $c_h \rightsquigarrow (\bigvee_{i=1}^{m} (\bigwedge_j c_{i_j}))$ can be interpreted as a material implication: if condition c_h is satisfied, then also $\bigvee_{i=1}^{m} (\bigwedge_j c_{i_j})$ must be satisfied. For instance, dependency d_1 states that if the storage server is located in France, the storage space is limited to 100TB (d_1), and dependency d_2 states that if the storage server is located in Italy, data are encrypted with the AES cypher (d_2).

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, the set of conditions to be included in the SLA must: *i)* satisfy $\mathcal{R}$; *ii)* satisfy D; and *iii)* be *well-formed*, that is, include at most one condition over each attribute $a \in A$. A set of conditions satisfying these three constraints is referred to as *valid SLA* (vSLA). With reference to our running example, a valid SLA could include the conditions ⟨loc=FR⟩, ⟨encr=AES⟩, ⟨max_storage=100TB⟩.

To compute a vSLA, the proposal in [1] introduces an assignment function $f : C \rightarrow \{0, 1\}$ assigning value 1 or value 0 to the conditions in C. Intuitively, all conditions c_i such that $f(c_i) = 1$ are the conditions forming a valid SLA. By using f to denote also the list of values assigned by f to the conditions in C (hence, $f(\mathcal{R})$ denotes the result of the evaluation of $\mathcal{R}$ with respect to f), the problem of determining a vSLA can be interpreted as finding a value assignment f such that: *i)* $f(\mathcal{R}) = 1$; *ii)* $f(d) = 1, \forall d \in D$; and *iii)* $\{c_i \in C : f(c_i) = 1\}$ is well-formed.

The proposal in [1] translates the problem of computing a vSLA in a Constraint Satisfaction Problem (CSP) [3], which can then be solved by adopting any CSP solver. 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 $w : X \rightarrow Z$ that satisfies all the constraints in K. Therefore:

- X includes all conditions appearing in $\mathcal{R}$ and in the set D of dependencies;
- Z corresponds to the set $\{0, 1\}$;
- K includes the requirement $\mathcal{R}$, the set D of dependencies, and the conflicts among conditions ensuring the set of conditions to be well-formed.

Intuitively, a valid SLA will include all conditions assigned value 1 by the assignment function w.

III. Supporting preferences

Given an application requirement $\mathcal{R}$ over a set C of conditions and a set $D = \{d_1, \ldots, d_l\}$ of dependencies over C, there can be different value assignments $f_1, \ldots, f_z$ that represent correct solutions to the problem of computing a vSLA. In fact, since an application requirement $\mathcal{R}$ is expressed as a DNF formula over a set of conditions, an assignment f_i represents a vSLA when it assigns value 1 to all conditions appearing in at least one of the conjunctive clauses in $\mathcal{R}$.

While all solutions $f_1, \ldots, f_z$ are equivalent in terms of correctness, the user might prefer one over another, depending on which clause is satisfied. For instance, with reference to our running example, the manager might prefer to maintain data stored in Italy rather than abroad. While a SLA only including $c_1 : \langle \texttt{loc=}IT \rangle$ and a SLA including $c_2 : \langle \texttt{loc=}FR \rangle$ and $c_5 : \langle \texttt{encr=}AES \rangle$ are both valid, the manager would prefer the first one. We now illustrate how preferences among alternative requirements can be expressed and considered in the computation of a vSLA.

Following our DNF formulation of requirements, we assume user preferences to be expressed as an ordering relationship $\succ$ among the different clauses k_i in $\mathcal{R}$, where $k_i \succ k_j$ iff a solution satisfying k_i is preferred to another solution satisfying k_j. For simplicity, we assume the existence of a single total order relationship with the note that our approach can be extended to accommodate multiple and/or partial orderings. With reference to our running example, $(c_1) \succ (c_2 \wedge c_5) \succ (c_4 \wedge c_6 \wedge c_7)$. Given an application requirement $\mathcal{R}$ and a preference defined over its clauses, a *preference-supporting* vSLA (pvSLA) is a vSLA that satisfies the most preferred clause. We can then define the problem of finding a pvSLA, as follows.

Problem III.1 (Preference-supporting 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, find a value assignment $f : C \rightarrow \{0,1\}$ such that:

1) *f represents a vSLA; and*
2) *$\nexists f'$, with f' a vSLA, such that $\exists k_i, k_j \in \mathcal{R}$, $k_j \succ k_i$, and $f'(k_j) = 1$, $f(k_j) = 0, f(k_i) = 1$.*

A solution to Problem III.1 is therefore represented by an assignment function f assigning value 1 to the conditions in the most preferred clause in $\mathcal{R}$ that can be part of a vSLA. For instance, the solution illustrated in Section II (i.e., $\langle$`loc`=FR$\rangle$, $\langle$`encr`=AES$\rangle$, $\langle$`max_storage`=100TB$\rangle$) is a vSLA but is not a pvSLA, as it satisfies $c_2 \wedge c_5$ but not c_1, which could be part of a vSLA and is preferred over $c_2 \wedge c_5$.

With the CSP-based formulation of the problem proposed in [1], a naive solution to select the most preferred solution among a set of possibilities (i.e., a pvSLA among the set of vSLAs) would consist in having the CSP solver enumerating all possible solutions, and then selecting the most suitable according to the user preference. While this approach would certainly do, it would require to explore the entire solution space. To avoid this, we allow users to specify their preferences along with the requirement $\mathcal{R}$, and to take them into account during the computation of a solution. The adoption of the `CLP(fd)` Prolog library can nicely help in the computation of a pvSLA, by allowing users to specify the order in which the variables of the CSP are grounded through the predicate `labeling` [4]. Such predicate is used to find a single solution with all ground variables and, when used with a `min` (`max`, resp.) option, causes the solution to be computed with the variable in the leftmost `min` (`max`, resp.) field considered first and grounded to their minimum, (maximum, resp.) value of the domain, going then down the list [4]. Since we aim at setting conditions to value 1, by evaluating predicate `labeling` with option $\texttt{max}(c_i), \ldots, \texttt{max}(c_j)$, the CSP solver will then try to find a solution assigning value 1 to conditions $c_i, \ldots, c_j$. A pvSLA is then computed through an iterative process: the CSP solver explores the solution space starting from the most preferred solution and if a a solution satisfying the most preferred clause does not exist, the solver is invoked looking for a solution satisfying the second most preferred clause, and so on until all clauses have been considered (according to the user preference over clauses) or a pvSLA is found. We note that, if a solution satisfying the `max` option of the `labeling` predicate is not found, the solver will return a solution (if it exists), as `false` is returned only when there are no (more) solutions to compute. This observation, coupled with our iterative approach, introduces a double advantage. First, if a solution satisfying the most preferred clause exists, it is found at the first iteration and, if no solution exists at all, it is immediately discovered. Second, since the solver will always return a solution (if it exists), the user can decide whether to accept such a solution (even it does not satisfy the preference) and terminates the iterative process, or to proceed in the iteration, checking whether there exists a solution for a less preferred clause. With reference to our running example, since $(c_1) \succ (c_2 \wedge c_5) \succ (c_4 \wedge c_6 \wedge c_7)$, the first iteration invokes the CSP solver with predicate `labeling` and option $\texttt{max}(c_1)$, returning a solution assigning value 1 to conditions c_1 and c_5 (c_5 being included in the SLA due to the triggering of dependency d_2). Such an assignment represents a vSLA and is also preference-supporting (Problem III.1), since it satisfies the most preferred clause that can be included in a SLA.

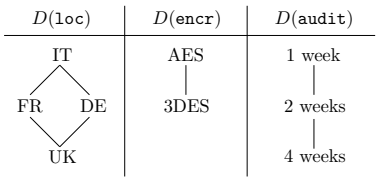

Fig. 2. Hasse diagrams representing the domains of attributes `loc`, `encr`, and `audit` with their value preferences

IV. Deriving preferences from attribute values

In the previous section, we have illustrated how preferences among alternative clauses of an application requirement can be specified by users and considered in the establishment of a SLA. However, it might be simpler to specify preferences over the *values* that can be assumed by the attributes characterizing the service provisioning rather than on the clauses themselves. We note that such a strategy might be useful for users who can have the need of moving to the cloud several applications with different requirements. In fact, they can specify their preferences over the values that can be accepted in the service provisioning once and for all, and then automatically derive preferences over the clauses based on them (thus saving the burden of manually specifying preferences among clauses for each application). For instance, with reference to our running example, a user might state that she prefer AES over 3DES since AES is more secure than 3DES.

We now illustrate how preferences among clauses can be possibly derived from preferences over attribute values. For simplicity, we focus our discussion on *equality* conditions of the form $c : \langle a = v \rangle$ with the note that on finite domains other conditions can be represented with them. For instance, with respect to our running example, condition $\langle$`audit` $\neq$ 1week$\rangle$ can be interpreted as $\langle$`audit` $=$ 2weeks$\rangle \vee \langle$`audit` $=$ 4weeks$\rangle$.

Given an attribute a, the preferences among the values of $D(a)$, called *value preferences* translate to a partial order relationship, denoted $\succ_a$, over the values in $D(a)$ such that, given two values $v_1, v_2 \in D(a), v_1 \succ_a v_2$ iff v_1 is *preferred* over v_2. For instance, given attribute `encr` with $D($`encr`$) = \{$AES,3DES$\}$, we have that AES$\succ_{encr}$3DES. Value preferences over the domain of an attribute can be naturally represented through a Hasse diagram. Figure 2 reports such graphical representation for attributes `loc`, `encr`, and `audit`, where values are represented as nodes and preferences over values are represented as edges among them. For instance, the preferences for `loc` state that value IT is preferred over FR and DE, which are in turn preferred to UK. We assume each domain to have a top value $v^\top$ (most preferred) and a bottom value $v^\bot$ (less

D(loc)	D(encr)	D(audit)
w(IT): 1 w(FR): 2/3 w(DE): 2/3 w(UK): 1/3	w(AES): 1 w(3DES): 1/2	w(1week): 1 w(2weeks): 2/3 w(4weeks): 1/3

Fig. 3. An example of score values

preferred). For instance, values IT and UK are the maximum and minimum values in $D(\texttt{loc})$ (see Figure 2).

Since we are considering equality conditions, value preferences can easily be used to derive the preferences over the clauses of an application requirement as follows. Intuitively, for each clause in an application requirement $\mathcal{R}$ we derive a *score vector* representing "how much" the user prefers the clause. To this purpose, we first assign a *score value* $w(v)$ to each value v in $D(a)$. Score $w(v)$ quantifies how v is ranked in the preferences over $D(a)$. In particular, we assume that score values are normalized in $(0,1]$ where, the higher the score, the more preferred the value. Using the Hasse diagram, we assign to value v a score $w(v)$ representing the relative depth of v in the diagram (i.e., with $\texttt{val}(v_i,v_j) = \texttt{val}(v_j,v_i)$ the number of values in the shorted path between v_i and v_j including them, $w(v) = \texttt{val}(v,v^\top)/\texttt{val}(v^\bot,v^\top)$). Note that other strategies for assigning these score values can work as well. Figure 3 illustrates the scores assigned to the values in the domains of attributes loc, encr, and audit.

Let $A^{\mathcal{R}} \subseteq A$ be the set of attributes involved in the conditions of an application requirement $\mathcal{R}$. We then associate each clause k in $\mathcal{R}$ with a *score vector* $\overrightarrow{W_k}$, with $|\overrightarrow{W_k}| = |A^{\mathcal{R}}|$ and $\overrightarrow{W_k}[a_i] = w(v_j)$, where v_j is the value required by the condition in k for attribute a_i. Note that the clauses in an application requirement $\mathcal{R}$ can include conditions on different (and possibly disjoint) sets of attributes. For instance, the first clause of the requirement of our running example includes only one condition c_1 on attribute loc, while the second clause k_2 includes two conditions c_2 and c_5 on attributes loc and encr, respectively. We assume that the lack of a condition over an attribute a in a clause k can simply mean that *any* value v in $D(a)$ is acceptable for the user. For instance, considering the requirement of our running example, since the first clause k_1 does not include any condition over attribute, say, encr, then *any* value in $D(\texttt{encr})$ would be fine for the user – as otherwise k_1 would also have included a condition imposing encr to assume a certain value. Therefore, given an attribute $a \in A^{\mathcal{R}}$ not involved in a clause k_i of a requirement $\mathcal{R}$, the score value w for a in the score vector $\overrightarrow{W_{k_i}}$ will be that of the most preferred value $v^\top$ in $D(a)$. With reference to our running example, consider clause k_2, which does not have any condition on attribute audit. The score vector $\overrightarrow{W_{k_2}}$ would then be defined as $\overrightarrow{W_{k_2}} = [2/3,1,1]$, assigning value 1 as the score for the (missing) value for attribute audit. The problem is now how to compare the score vectors associated with the different clauses in $\mathcal{R}$ to derive their preferences. A simple approach consists in summing the score values in the vectors of the clauses, resulting in a *clause score* (since score values for all attributes are normalized, the same score assigned to values of two different attributes represent the same ranking in the Hasse diagrams of the two attributes). The preferences over clauses would then be immediately derived from the ordering of the clauses based on such score. To illustrate, consider our running example. The score vectors of the three clauses k_1, k_2, and k_3 of $\mathcal{R}$ are $\overrightarrow{W_{k_1}} = [1,1,1]$, $\overrightarrow{W_{k_2}} = [2/3,1,1]$, and $\overrightarrow{W_{k_3}} = [1/3,1/2,1]$, respectively. The clause score assigned to k_1 (k_2 and k_3, resp.) is therefore 3 ($8/3$ and $11/6$, resp.). Since $3 > 8/3 > 11/6$, the resulting preference among clauses is defined as $k_1 \succ k_2 \succ k_3$ (which corresponds to $(c_1) \succ (c_2 \wedge c_5) \succ (c_4 \wedge c_6 \wedge c_7)$). A pvSLA satisfying such preference is the SLA illustrated in Section III, which includes $c_1 : \langle \texttt{loc} = \text{IT} \rangle$ and $c_5 : \langle \texttt{encr} = \text{AES} \rangle$.

V. Related work

The appeal of cloud computing has attracted the attention of researchers who have been investigated solutions for addressing several challenging issues, ranging from the protection of the confidentiality and integrity of data and computations, fault tolerance, private access, and the specification and assessment of requirements to be satisfied by providers (e.g., [1], [5], [6], [7], [8], [9]). In particular, the work presented in this paper extends the proposal in [1] with the consideration of user-based preferences in the definition of a SLA. The original proposal in [1] was framed in the context of cloud-based IoT information processing where the adoption of cloud solutions to store and manage IoT information has been recently subject of several proposals (e.g., [10], [11], [12]). Our work, also applicable to cloud-based processing of IoT information, addresses an issue orthogonal to such works.

Another close line of work is related to the problem of establishing a SLA between users and cloud providers. The solution in [13] focuses on the assessment of services offered by cloud providers based on customer requirements. Similarly, also the work in [14] aims at comparing different cloud providers over different performance indicators, focusing specifically on the performance and cost of the providers. The goal of this proposals is however that of prioritizing cloud services, while we focus on defining and enforcing preferences over user requirements in the computation of a SLA. The proposal in [15], while providing 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, and does not model preferences over the requirements. Multi-criteria decision making techniques have also been extensively adopted to rank / prioritize cloud providers (e.g., [16], [17]), considering, however, a scenario different from ours, characterized by the existence of an application requirement as in [1], and aimed at prioritizing the alternative options included in the requirement.

VI. Conclusions

We have presented an approach for allowing users to define their preferences over different alternative clauses in their application requirements. We have also shown how such

preferences can be considered in the definition of a SLA meeting the users' requirements, and how such preferences can be derived from preferences expressed over the values of the properties characterizing a cloud service.

ACKNOWLEDGMENTS

This work was supported in part by the EC within the 7FP under grant agreement 312797 (ABC4EU) and within the H2020 under grant agreement 644579 (ESCUDO-CLOUD).

REFERENCES

[1] S. De Capitani di Vimercati, G. Livraga, V. Piuri, P. Samarati, and G. Soares, "Supporting application requirements in cloud-based iot information processing," in *Proc. of the International Conference on Internet of Things and Big Data (IoTBD 2016)*, Rome, Italy, April 2016.

[2] M. Galster and E. Bucherer, "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, July 2008.

[3] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati, "Fragmentation in presence of data dependencies," *IEEE Transactions on Dependable and Secure Computing (IEEE TDSC)*, vol. 11, no. 6, pp. 510–523, November/December 2014.

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

[5] S. De Capitani di Vimercati, S. Foresti, and P. Samarati, "Managing and accessing data in the cloud: Privacy risks and approaches," in *Proc. of the 7th International Conference on Risks and Security of Internet and Systems (CRiSIS 2012)*, Cork, Ireland, October 2012.

[6] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati, "Efficient and private access to outsourced data," in *Proc. of the 31st International Conference on Distributed Computing Systems (ICDCS 2011)*, Minneapolis, Minnesota, USA, June 2011.

[7] R. Jhawar and V. Piuri, "Fault tolerance management in iaas clouds," in *Proc. of the 2012 IEEE Conference in Europe about Space and Satellite Telecommunications (ESTEL 2012)*, Rome, Italy, October 2012.

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

[9] R. Jhawar and V. Piuri, "Fault tolerance and resilience in cloud computing environments," in *Computer and Information Security Handbook, 2nd Edition*, J. Vacca, Ed. Morgan Kaufmann, 2013, pp. 125–141.

[10] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu, "An IoT-oriented data storage framework in cloud computing platform," *IEEE Transactions on Industrial Informatics (IEEE TII)*, vol. 10, no. 2, pp. 1443–1451, May 2014.

[11] Y. Ma, J. Rao, W. Hu, X. Meng, X. Han, Y. Zhang, Y. Chai, and C. Liu, "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, October-November 2012.

[12] M. Kovatsch, M. Lanter, and Z. Shelby, "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, October 2014.

[13] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012–1023, Jun. 2013.

[14] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: Comparing public cloud providers," in *Proc. of ACM SIGCOMM 2010*, Melbourne, Australia, August – September 2010.

[15] R. Jhawar, V. Piuri, and P. Samarati, "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, December 2012.

[16] Z. Rehman, O. Hussain, and F. Hussain, "IaaS cloud selection using MCDM methods," in *Proc. of the 9th IEEE International Conference on e-Business Engineering (IEEE ICEBE 2012)*, Hangzhou, China, September 2012.

[17] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012–1023, 2013.