Structural Optimization of Reduced Ordered Binary Decision Diagrams for SLA Negotiation in IaaS of Cloud Computing*

Kuan Lu[1], Ramin Yahyapour[1], Edwin Yaqub[1], and Constantinos Kotsokalis[2]

[1] Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany
[2] IT & Media Center of Dortmund University of Technology, Germany
{kuan.lu,ramin.yahyapour,edwin.yaqub}@gwdg.de,
constantinos.kotsokalis@tu-dortmund.de

Abstract. In cloud computing, an automated SLA is an electronic contract used to record the rights and obligations of service providers and customers for their services. SLA negotiation can be a time-consuming process, mainly due to the unpredictable rounds of negotiation and the complicated possible dependencies among SLAs. The operation of negotiating SLAs can be facilitated when SLAs are translated into Reduced Ordered Binary Decision Diagrams (ROBDDs). Nevertheless, an ROBDD may not be optimally structured upon production. In this paper, we show how to reduce the number of 1-paths and nodes of ROBDDs that model SLAs, using ROBDD optimization algorithms. In addition, we demonstrate the reduction of 1-paths via the application of Term Rewriting Systems with mutually exclusive features. Using the latter, ROBDDs can be generated accurately without redundant 1-paths. We apply the principles onto the negotiation of IaaS SLAs via simulation, and show that negotiation is accelerated by assessing fewer SLA proposals (1-paths), while memory consumption is also reduced.

Keywords: Cloud computing, IaaS, SLA negotiation, Term rewriting, ROBDD structural optimization.

1 Introduction

Recent years have witnessed wide adoption of utility computing for service provisioning, in favor of more adaptive, flexible and simple access to computing resources. Utility computing has enabled a pay-as-you-go consumption model for computing similar to traditional utilities such as water, gas or electricity [1]. As a realization of utility computing [2], cloud computing provides computing utilities that can be leased and released by the customers through the Internet in an on-demand fashion. More recently, from the perspective of service type, three service delivery models are commonly used, namely, *Software-as-a-Service*

* The research leading to these results is supported by Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG) in Germany.

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 268–282, 2012.

(SaaS), *Platform-as-a-Service* (PaaS) and *Infrastructure-as-a-Service* (IaaS) [5]. Automated negotiation may be used to accommodate a customer's heterogeneous requirements against a service provider's capabilities and acceptable usage terms. The result of such a negotiation is a Service Level Agreement (SLA), an electronic contract that establishes all relevant aspects of the service. SLA negotiation can be *time-consuming*, mainly due to the unpredictable rounds of negotiation and the possible complicated dependencies among SLAs. For instance, an SaaS provider negotiates with one or more IaaS providers for computing resources to host her applications. Thus, a SaaS SLA could have dependencies with one or more IaaS SLAs. Establishing an SLA might need one or more rounds of negotiation until both sides agree with the stipulation of the contract.

In our previous work [20], SLAs were canonically represented as Reduced Ordered Binary Decision Diagrams (ROBDDs), aiming to *facilitate* SLA negotiation. A BDD diagram includes some decision nodes and two terminal nodes (0-terminal and 1-terminal). A path from the root node to the 1-terminal represents a variable assignment for which the Boolean function is true. Such a path is called "1-path". A 1-path can also be treated as an SLA proposal. Nevertheless, ROBDDs may be suboptimal in structure. Firstly, the ROBDDs should be maintained throughout the whole life cycle of SLAs, so diagrams with too many nodes may waste a lot of memory. Secondly, an ROBDD may have semantically redundant 1-paths that improperly reflect a customer's requirements; as a result, SLA negotiation may slow down.

In this paper, we propose that by applying BDD node optimization algorithms, the number of nodes for the ROBDDs can be decreased efficiently. Thus, SLAs occupy less memory space. Additionally, the size and number of paths (especially as regards 1-paths) can be reduced by eliminating those that are semantically redundant via BDD path optimization algorithms. Consequently, the SLA negotiation is accelerated by assessing fewer SLA proposals (1-paths). Furthermore, a novel alternative solution for reducing semantically redundant 1-paths during construction is introduced. We argue that all the options of an SLA term are mutually exclusive. Thus, customizing an SLA term is rewritten as correctly selecting an option for this term. This process can be treated as an implementation of an SLA *Term Rewriting System*.

The remainder of this paper is structured as follows. In Section 2, we discuss related work. Section 3 describes how an SLA can be modeled as an ROBDD. In Section 4, we present how the structure of an initial ROBDD can be optimized. In order to validate our mechanisms, in Section 5, we simulate the SLA negotiation by transforming IaaS SLAs into ROBDDs and optimizing the initial ROBDDs. Finally, we conclude the paper in Section 6.

2 Related Work

In cloud computing, SLAs are used to ensure that service quality is kept at acceptable levels [8]. An SLA management component may consist of several components: discovery, (re-)negotiation, pricing, scheduling, risk analysis, monitoring, SLA enforcement and dispatching [7], where SLA negotiation determines

the cardinality of parties involved, their roles, the visibility of the offers exchanged and so on [30]. Furthermore, Ron et al. in [9] define the SLA life cycle in three phases: creation phase, operation phase and removal phase. Our focus is on IaaS SLA negotiation during the SLA creation phase.

Based on [7], our framework [3] [4] [20] (developed in the SLA@SOI EU Integrated Project [29]) covers the features for multi-round negotiation with counteroffers customer and service provider. In general, apart from IaaS, the framework can be easily extended to other scenarios (e.g., SaaS, PaaS) as well in which automated SLA management is considered. In our scenario, virtual resources are created and provisioned through Open Cloud Computing Interface (OCCI) [25] and OpenNebula [26]. The VMs are compute instances, connected with OS image and network instances. The storage pool is formed by several images, which means that customers are able to uploade their own images.

The service provider can abstract the services in perspicuous terms, e.g., the instances of Amazon EC2 [21]. Thus, non-technical customers can mainly focus on the service as a whole rather than considering the service in too much detail. However, the service provider would best allow customers with flexibility and adaptability [22]. Chandra et al. [23] suggest that fine-grained temporal and spatial resource allocation may lead to substantial improvements in capacity utilization. Therefore, in IaaS, this means that customers are free to customize the VM configuration. Namely, the granularity of customizing VMs evolves from setting the number of predefined VM to the detailed specification of each term in a VM. In this paper, the SLA terms that can be customized (see Table 1), the ROBDD structural optimization in Section 4 and the simulation in Section 5 are based on the cloud infrastructure provided by the GWDG for its customers and scientific communities.

Service availability is one of the most important Quality of Service (QoS) metrics with respect to IaaS. In [24], authors outline that the service availability

Table 1. SLA terms and descriptions

SLA term	**Option**	**Variable**
Service_name	[service name]	[x_1]
Business_hours	[09:00-17:00]	[x_2]
VM_number	[1]	[x_3]
CPU_core	[1, 2, 4]	[x_4, x_5, x_6]
CPU_speed	[2.4 GHz, 3.0 GHz]	[x_7, x_8]
Memory	[1 GB, 2 GB]	[x_9, x_{10}]
Network	[Net-1, Net-2]	[x_{11}, x_{12}]
Storage_image	[Private, OS-1, OS-2]	[x_{13}, x_{14}, x_{15}]
Service_availability	[99.99%, 99.9%]	[x_{16}, x_{17}]

guaranteed by three large cloud providers (Amazon, Google and Rackspace Cloud) is more than 99.9% in order to obtain good reputation in todays competitive market. Therefore, we propose to provide a basic service availability of 99.9% and an advanced availability of 99.99%. The later one implies that the service provider has to pay special attention (e.g., extra resources) on the service during SLA monitoring phase in order to avoid SLA violation.

At the moment, many representations of SLAs, e.g., WS-Agreement [16], WSLA [17] and the SLA model in [19], focus on enabling interoperability between independent agents. However, our focus is on a system-internal representation that is able to efficiently support decision-making during SLA negotiation. In [20], we presented a novel application of ROBDD, for representing, managing and facilitating the construction of SLAs. During BDD construction, although pushing all facts and conditions to the top of the diagram provides a possibility for optimizing the BDD, there could still be lots of semantically redundant 1-paths. Furthermore, this kind of ordering does not reduce the total number of decision nodes, which waste memory space.

In order to optimize the structure of the ROBDD, we studied Term Rewriting Systems (TRS) for Boolean functions [14] [15] that could be applied in SLA terms selection to reduce the number of redundant 1-paths of an ROBDD. In mathematics, rewriting systems cover a wide range of methods of transforming terms of a formula with other terms. In TRS, a term can be recursively defined to a constant c, or a set of variables $x_1 ... x_n$, or a function f [15]. The terms are composed of binary operators logical conjunction "$\wedge$", logical disjunction "$\vee$" and unary operator "$\neg$". In IaaS, an SLA term contains one or more variables, namely the options. We make an effort to rewrite the set of SLA terms while customizing SLA templates in a way that depicts the customer's request precisely.

Alternatively, the existing BDD optimization algorithms in [10] [12] [28] [31] provide us with the theoretical foundation to reduce the size of ROBDD. Furthermore, JavaBDD [13], a Java library for manipulating BDDs, is the tool we chose for setting up the BDD handling and programming environment.

3 Modeling SLA with ROBDD

An SLA is essentially a set of facts and a set of rules. Facts are globally (with respect to the contract) applicable truths, such as parties involved, monetary unit, etc. Rules include:

- the conditions that must hold for a certain clause to be in effect;
- the clause, typically describing the expected result that the customer wishes to receive and which is usually referred to as Service Level Objective (SLO);
- a fall-back clause in the case that the aforementioned clause is not honored.

As an example, for the condition "time of day is after 08:00", the clause could be "service availability $\geq$ 99.9%", and the fall-back clause could be an applicable penalty. This kind of format actually reflects real-life contracts and their if-then-else structure.

BDDs, based on Shannon's decomposition theorem [6], are well-known in the domain of Computer Aided Design (CAD) for Very Large Scale Integrated (VLSI) circuits. They can represent Boolean functions as rooted, directed and acyclic graphs, which consist of decision nodes and two terminal nodes called 0-terminal and 1-terminal. Each decision node is labeled by a Boolean variable and has two child nodes called low child and high child. A path from the root node to the 1-terminal represents a variable assignment for which the Boolean function is true. Such a path is also called "1-path". Compared to other techniques to represent Boolean functions, e.g., truth tables or Karnaugh maps, BDDs often require less memory and offer faster algorithms for their manipulation [10].

A BDD, with all variables occurring in the same order on all paths from the root, is said to be ordered (OBDD). Furthermore, if all identical nodes are shared and all syntactically redundant paths are eliminated, the OBDD is said to be reduced, shortly termed ROBDD [12]. For example, Eq. (1) is a disjunction Boolean function with 3 variables. Its corresponding ROBDD is illustrated in Fig. 1 (a), including 3 decision nodes (x_1, x_2, x_3) and 3 1-paths (x_1 = true, x_2 = false, x_3 = false), (x_1 = false, x_2 = true, x_3 = false) and (x_1 = false, x_2 = false, x_3 = true). As already mentioned, ROBDDs are useful for modeling SLAs due to their capability to provide canonical representations generated on the grounds of if-then-else rules. Especially, ROBDDs can express SLAs unambiguously. Equivalent SLAs, which are structurally different, are eventually represented by the same ROBDD. On the contrary, using formats developed for on-the-wire representation such as WS-Agreement [16] does not guarantee this property. Hence, ROBDDs can be used internally in systems that have to manage SLAs.

$$f(x_1, x_2, x_3) = x_1 \vee x_2 \vee x_3 \tag{1}$$

An SLA could have dependencies with one or more sub-SLAs. The SLA manager has to parse the SLA request into an ROBDD and analyze all of its 1-paths. A 1-path is a potential SLA proposal that satisfies the customer's requirements. Nevertheless, an ROBDD may have semantically redundant 1-paths that reflect the customer's requirements improperly and reduce the time efficiency of the negotiation process. Moreover, the SLA manager should maintain the ROBDDs throughout the whole life cycle of SLAs. Accordingly, the size of 1-path and node of ROBDDs for SLAs ought to be simplified and controlled.

4 ROBDD Structural Optimization

4.1 SLA Term Rewriting System with Mutual Exclusiveness in ROBDD

According to the canonicity Lemma in [12], by reducing all the syntactically redundant paths, there is exactly one ROBDD with n basic variables in the order of $x_1 < x_1 < ... < x_n$. In this unique ROBDD, all variables of a disjunction function are mutually exclusive. Based on the truth table, e.g., a disjunction function is true when all its variables are true. However, such an assignment does

not exist in ROBDD; as the ROBDD checks these variables one after another, the first true variable already ensures this function to be true and that leaves the rest of the variables not evaluated. For instance, as we explained in Section 3, the ROBDD (Fig. 1 (a)) of Eq. (1) has 3 decision nodes and 3 1-paths and its structure can't be simplified anymore. Therefore, if the inputs of a disjunction function are the basic variables, they are mutually exclusive and the ROBDD contains no redundant nodes and 1-paths. In contrast, when the inputs are not basic variables, despite the mutually exclusive feature among the inputs (only one of the inputs will be selected), the ROBDD of this disjunction function might have redundant nodes and 1-paths. *Semantically, options of most SLA terms are mutual exclusive, which means the customer can only choose one of them and this term must be represented as a combination of all the options with binary operators "logical conjunction" (∧), "logical disjunction" (∨) and unary "negation" (¬).* While constructing an ROBDD, using Table 1, a simple SLA (see Eq. (2)) can be customized by specifying SLA term "Network" with 2 mutually exclusive options "x_{11}" and "x_{12}" and SLA term "Service_availability" with 2 mutually exclusive options "x_{16}" and "x_{17}". This SLA indicates that inputs "$x_{11} \wedge x_{16}$" and "$x_{12} \wedge x_{17}$" both are acceptable for the customer.

$$SLA = (x_{11} \wedge x_{16}) \vee (x_{12} \wedge x_{17}) \tag{2}$$

Its ROBDD (see Fig. 1 (b)) includes 6 decision nodes and 4 1-paths. However, two of those paths, namely (x_{11} = true, x_{12} = true, x_{16} = false, x_{17} = true) and (x_{11} = true, x_{12} = true, x_{16} = true), are not correct, since x_{11} and x_{12} cannot be true concurrently. In IaaS, selecting both "Net-1" and "Net-2" as network is an illogical clause. Therefore this inaccurate SLA representation creates two unrealistic semantically redundant 1-paths and such paths should be eliminated at the very beginning.

Consequently, we propose that specifying an SLA term (t) with 2 options (α_1 and α_2) can be rewritten as illustrated in Formula (3).

$$t \rightarrow (\alpha_1 \wedge \neg\alpha_2) \vee (\neg\alpha_1 \wedge \alpha_2) \rightarrow \alpha_1 \oplus \alpha_2 \tag{3}$$

Mutual exclusiveness cannot be simply represented as a combination of all the options with the exclusive disjunction "⊕", when there are more than 2 options in an SLA term. Because the output is true when the number of "true" variables is odd and the output is false when the number of "true" variables is even [27]. For example, all-true assignment makes the expression $\alpha_1 \oplus \alpha_2 \oplus \alpha_3$ true, which is however not what we expect. Thus, when an SLA term (t) contains n ($n \geq 3$) options (α_1,..., α_n), we have the following assumptions:

$$N = \{1, ..., n\} \tag{4}$$

$$A \cup B = N \tag{5}$$

$$A \cap B = \emptyset \tag{6}$$

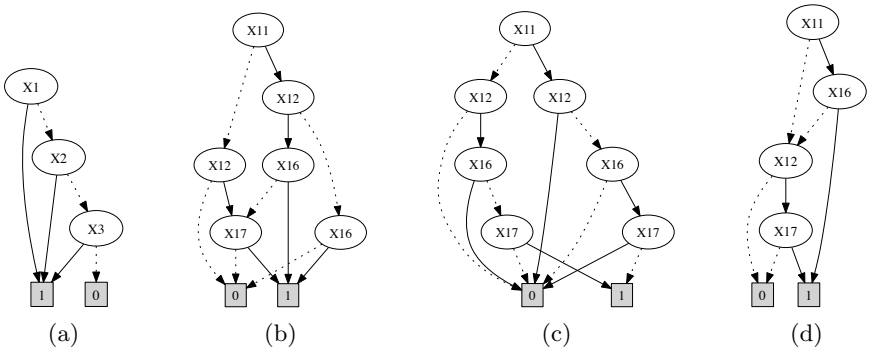


Fig. 1. (a) The ROBDD of disjunction function, (b) Mutual exclusiveness of disjunction function but with semantically redundant 1-path in ROBDD, (c) Mutual exclusiveness of disjunction function without semantically redundant 1-path in TRS ROBDD, (d) ROBDD after path optimization

$$A \neq \emptyset, B \neq N \tag{7}$$

From Eq. (4) to (6), N is a set of sequential numbers of all options. N can be further divided into 2 disjoint subsets. Options that concern a customer are put into A and indifferent ones are in B. Eq. (7) means that the customer should select at least one option. Thereby specifying an SLA term (t) with n ($n \geq 3$) options can be rewritten as Eq. (8).

$$t \rightarrow \bigvee_{l \in A} \left(\left(\bigwedge_{k \in A} \neg \alpha_k \right) \wedge \alpha_l \right) \wedge \left(\bigwedge_{m \in B} \neg \alpha_m \right), l \neq k \neq m \tag{8}$$

For SLA term t, $\bigcup_{l \in A} a_l$ is a set of options that the customer is flexible with and the rest of the options in A ($\bigcup_{k \in A} a_k$) should be denied *explicitly* with "¬". In the meanwhile, the unconcerned options in N, namely set B, should also be negated with "¬" in order to not conflict with other SLA option(s) from the same customer. Therefore, when $n = 3$, Eq. (8) can be specified as Eq. (9), in this case the set B is empty, which means the customer is flexible with any option of α_1, α_2, α_3.

$$(\alpha_1 \wedge \neg\alpha_2 \wedge \neg\alpha_3) \vee (\neg\alpha_1 \wedge \alpha_2 \wedge \neg\alpha_3) \vee (\neg\alpha_1 \wedge \neg\alpha_2 \wedge \alpha_3) \tag{9}$$

Based on above concepts, Eq. (2) can be written as Eq. (10) with 7 decision nodes and 2 1-paths in its TRS ROBDD (see Fig. 1 (c)). The number of redundant 1-paths is reduced efficiently although the node size increases by 1.

$$(x_{11} \wedge x_{16} \wedge \neg x_{12} \wedge \neg x_{17}) \vee (x_{12} \wedge x_{17} \wedge \neg x_{11} \wedge \neg x_{16}) \tag{10}$$

In summary, the above term rewriting concepts can be set into a dictionary and updated dynamically according to the use case, whereby the semantically redundant 1-paths can be eliminated efficiently. This also reduces the complexity of planning and optimization processes for SLA management. However, the shortcoming is that perhaps this approach might introduce extra decision nodes.

4.2 ROBDD Variable Swap and Sifting Algorithm

Alternatively, structural optimization algorithms can be applied to reduce the size of ROBDD. Here, the size means the number of nodes or paths. The algorithms make use of basic techniques such as variable swaps and the sifting algorithm. As approved in [10], *a swap of adjacent variables in a BDD only affects the graph structure of the two levels involved in the swap, leaving the semantic meaning of Boolean function unchanged.*

Algorithm 1. Sifting algorithm [11]

```
sort level numbers by descending level sizes and store them in array sl;
for i = 1 → n do
  if sl[i] = 1 then
    sift_down(i, n); // the BDD size is recorded in sift_down();
  else if sl[i] = n then
    sift_up(i, 1); // the BDD size is recorded in sift_up();
  else if (sl[i] − 1) > (n − sl[i]) then
    sift_down(i, n);
    sift_up(n, 1);
  else
    sift_up(i, 1);
    sift_down(1, n);
  end if
  sift_back();
end for
```

Based on the variable swap, the classical sifting algorithm is described in Algorithm 1, where the levels are sorted by descending level sizes. The largest level contains the most nodes and is considered first. Then, the variable is moved downwards until the terminal nodes and upwards from the initial position to the top. In the previous steps, the BDD size resulting from every variable swap is recorded. In the end, the variable is moved back to the position, which led to a minimal BDD size. Here, the size could be the number of nodes or the 1-paths.

4.3 Node Optimization

The sifting algorithm, based on the efficient exchange of adjacent variables, is able to dynamically reorder the structure of BDD in a way to change the number of decision nodes. While the sifting algorithm is executing, we record the BDD node size for each variable swap. In the meanwhile, we also store all the $< node,\ 1-path >$ pairs into a $< node,\ 1-path >$ array, which can further be used in Section 4.5 for determining the optimal $< node,\ 1-path >$ pair. In the end, a BDD with minimum number of decision nodes is derived.

In Section 4.1, we strive to define the TRS ROBDD accurately enough so that no semantically redundant 1-paths exist. Thereby, we say *the quantity and semantic meaning of 1-path of TRS ROBDD do not vary with the changes of*

variable ordering of TRS ROBDD. As we showed that a TRS ROBDD might introduce extra decision nodes, we can further use node optimization to reduce its node size. Clearly, in this case node optimization only improves the node size and leaves the 1-path unchanged.

In-memory, each decision node requires an index and pointers to the succeeding nodes [28]. Since each decision node in an ROBDD has two pointers, the memory size required to represent an ROBDD is given by Eq. (11).

$$Memory(ROBDD) = (1 + 2) \times nodes(ROBDD) \tag{11}$$

4.4 Path Optimization

Apart from the BDD node optimization, another criterion –namely, the number of 1-paths– for BDD optimality is also considered. As the variable ordering heavily influences the number of nodes of a BDD, the sifting algorithm can be modified to minimize the number of 1-paths instead of the node size of a given BDD. After each swap of two adjacent variables, i.e. after processing all nodes in the two levels, changes are only propagated from those two levels down to the terminal nodes. During modified sifting no upper limit on the number of 1-paths is used to stop the swapping operations [10].

Similarly, we record the 1-paths number of BDD for each variable swap. In the meanwhile, we also store all the $< 1 - path,\ node >$ pairs into a $< 1 - path,\ node >$ array, which can further be used in Section 4.5 for determining the optimal $< 1 - path,\ node >$ pair. In the end, a BDD with minimum number of 1-paths is derived.

Although for Eq. (2), the 1-paths number can be reduced from 4 to 3, path (x_{11} = true, x_{12} = true, x_{16} = false, x_{17} = true) still exists (see Fig. 1 (d)), where x_{11} and x_{12} are true at the same time. Path optimization relieves the work of SLA management, but it does not eliminate the semantically redundant 1-paths completely. Thus, the SLA manager still needs to evaluate the validity of each 1-path despite the partial reduction of 1-paths.

4.5 Multicriteria Optimization Problem

As it is demonstrated in [10], the number of paths can be significantly reduced for some benchmarks. At the same time, the number of nodes does not necessarily increase and may even be reduced.

The path optimization algorithm re-constructs an ROBDD with the minimal number of 1-paths, but not necessarily the minimal number of decision nodes. Similarly, node optimization algorithm re-constructs the ROBDD with the minimal number of decision nodes, but not necessarily the minimal number of 1-paths. As Lemma 5.5 in [10], for all Boolean functions of two or three variables there exists a BDD that is minimal both in size and the number of 1-paths. This is however not true for functions of more than three variables. Therefore, this becomes a multicriteria optimization (MCO) problem for gaining a minimal number of decision nodes and 1-paths (see Algorithm 2).

Algorithm 2. Calculate optimal (node, path) pair

```
store node and 1_path size of node_minimization() into n_nopti and p_nopti;
store node and 1_path size of path_minimization() into n_popti and p_popti;
if n_nopti = n_popti then
   return (n_popti, p_popti);
else if p_nopti = p_popti then
   return (n_nopti, p_nopti);
else
   return node_path_pair_selection();
end if
```

In Algorithm 2, if the node size of an ROBDD after executing the path optimization is equal to that after executing the node optimization, the result of the path optimization will be taken. An analogous statement holds if the path size of an ROBDD after executing the node optimization is equal to that after executing the path optimization, thus we take the result of the node optimization. These two situations mean we can get minimal size of paths and decision nodes at the same time. Otherwise, we can't have an optimized ROBDD in both ways. A compromise between the two measures should be considered in *node_path_pair_selection*(). Here, end users have to specify it according with their requirements. For example, the customer may only concern the number of 1-paths regardless of number of nodes or the other way around. A sample solution based on geometric distance [32] for this MCO problem could be resolved by Eq. (12). Selection of a point that is closest to the Optimal Point (OP). A preference is given to the point with the smallest number of 1-paths, when multiple points have the same distance to the OP.

$$Distance\_to\_OP = \sqrt{(n\_opti)^2 + (p\_opti)^2} \tag{12}$$

5 Experimental Verification

Based on the SLA template (Table 1), we assume that a customer starts an SLA negotiation for service "IaaS-1", given that business hours are between 09:00 and 17:00. The customer needs one VM with 2 or 4 CPU cores, CPU speed is either 2.4 GHz or 3.0 GHz, memory size is 2 GB, network is 10 Gb/s Net-1, either OS-1 or OS-2 is selected and customer's private image is uploaded, service availability must be 99.99% or higher; *Or* a VM with 1 CPU core, CPU speed must be 2.4 GHz, memory size is 1 GB, network is 10 Gb/s either Net-1 or Net-2, either OS-1 or OS-2 is selected, no private image is uploaded, service availability is at least 99.9% or higher.

Table 2 illustrates the set of facts and clauses that we will use for this use case scenario. It is straightforward to see that these facts and clauses can be considered as Boolean variables, which evaluate to true or false. The SLA can also be correctly evaluated if it is modeled according to the following equations.

$$f_1 = x_1 \wedge x_2 \tag{13}$$

Table 2. Example clauses of an SLA template

Variable	Proposition	Proposition Type
x_1	Service_name = "IaaS-1"	Fact
x_2	Business_hours = 09:00 - 17:00	Fact
x_3	VM = "1"	Clause
$x_5 \vee x_6$	CPU_core = "2" or "4"	Clause
$x_7 \vee x_8$	CPU_speed = "2.4 or 3.0" GHz	Clause
x_{10}	Memory = "2 GB"	Clause
x_{11}	10 Gb/s Network = "Net-1"	Clause
x_{13}	Storage = "Private image"	Clause
$x_{14} \vee x_{15}$	Storage = "OS-1 image" or "OS-2 image"	Clause
x_{16}	Service_availability $\geq$ 99.99%	Clause
x_3	VM = "1"	Clause
x_4	CPU_core = "1"	Clause
x_7	CPU_speed = "2.4 GHz"	Clause
x_9	Memory = "1 GB"	Clause
$x_{11} \vee x_{12}$	10 Gb/s Network = "Net-1" or "Net-2"	Clause
$\neg x_{13}$	Storage != "Self image"	Clause
$x_{14} \vee x_{15}$	Storage = "OS-1 image" or "OS-2 image"	Clause
x_{17}	Service_availability $\geq$ 99.9%	Clause

$$f_2 = x_3 \wedge (x_5 \vee x_6) \wedge (x_7 \vee x_8) \wedge x_{10} \wedge x_{11} \tag{14}$$

$$f_3 = x_{13} \wedge (x_{14} \vee x_{15}) \wedge x_{16} \tag{15}$$

$$f_4 = x_3 \wedge x_4 \wedge x_7 \wedge x_9 \wedge (x_{11} \vee x_{12}) \tag{16}$$

$$f_5 = \neg x_{13} \wedge (x_{14} \vee x_{15}) \wedge x_{17} \tag{17}$$

$$SLA = f_1 \wedge ((f_2 \wedge f_3) \vee (f_4 \wedge f_5)) \tag{18}$$

This SLA request is firstly transformed to an initial ROBDD with 29 decision nodes and 40 1-paths. Memory requirement is 87 indices and pointers. By applying the term rewriting, the number of 1-paths of the initial ROBDD is reduced to be 12, however, the decision nodes are increased by 2 (31 decision nodes).

As already mentioned in Section 4.3, we can further optimize this ROBDD by using the node optimization. There will be 30 decision nodes in this ROBDD (see Fig. 2 (a)). Memory requirement is 90 indices and pointers. Alternatively, by applying the path and node optimization, the initial ROBDD is optimized to be with 21 decision nodes and 12 1-paths. Memory requirement is 63 indices and pointers. (see Fig. 2 (b)). Eventually, we attempted to simulate the similar SLA negotiation above for 1000 times to compare the performance of three approaches. We reused our planning and optimization algorithms in [3] to balance the price, profit and failure rate. Each time, the SLA template was customized randomly by selecting different combinations of options for all the SLA terms. Each SLA template was first transformed into an initial ROBDD using the approach in [20]. Then we rewrote the same initial ROBDD using TRS, and following that we used BDD node optimization to reduce the nodes of TRS ROBDD by the greatest extent. Finally, we optimized the initial ROBDD by using BDD optimization algorithms. All the decision nodes and 1-paths for each approach were aggregated and compared with each other. Additionally, we assumed that each round of negotiation starts when the customer submits the SLA template to SLA manager and stops when the SLA manager sends an offer or a counter-offer back to the customer. The total negotiation time of each approach

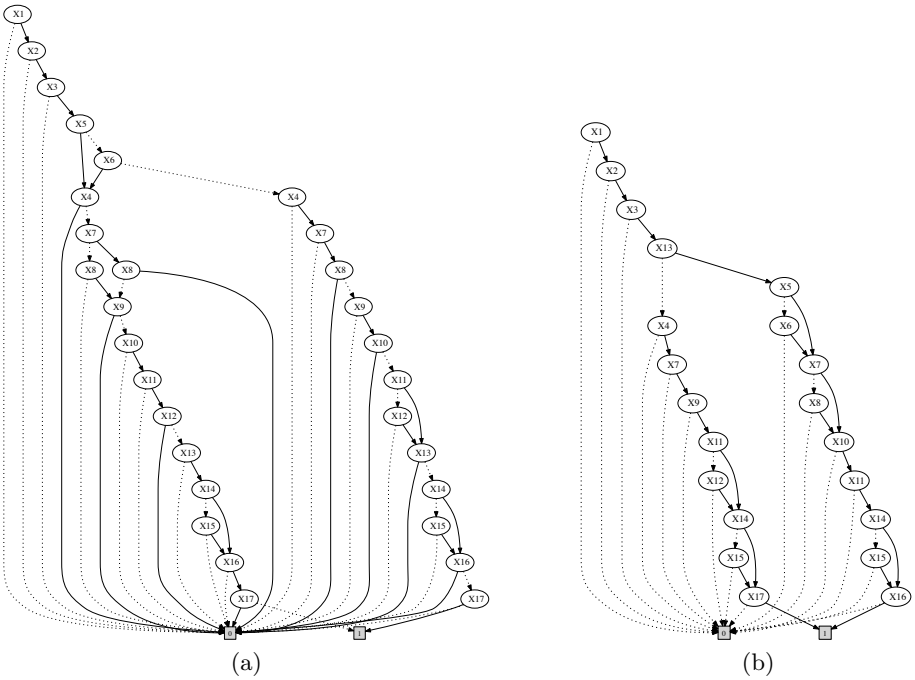
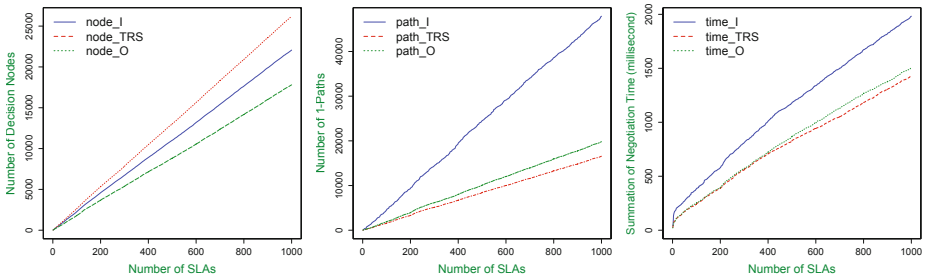


Fig. 2. (a) The ROBDD with 30 decision nodes and 12 1-paths by applying term rewriting and node optimization. (b) The ROBDD with 21 decision nodes and 12 1-paths by applying path and node optimization



(a) (b) (c)

Fig. 3. The number of nodes (a), 1-paths (b) and negotiation time (c) statistics of the initial ROBDD (blue), the TRS ROBDD (red) and the ROBDD after running BDD optimization algorithms (green)

was counted. Thus, the whole simulation took approximately 13521 seconds on a 1.8 GHz processor. Initial ROBDDs had 22123 decision nodes (*node_I*) and 47880 1-paths (*path_I*) and the negotiation time (*time_I*) was 1982 milliseconds (ms). Fig. 3 (a) illustrates that the number of decision nodes increased proportionally in all approaches. The TRS ROBDDs had the most decision nodes (*node_TRS*=26421), because the least number of 1-path (*path_TRS*=16554) in Fig. 3 (b) and the least negotiation time (*time_TRS*=1432 ms) in Fig. 3 (c) were realized at the cost of nodes. *Eclectically*, BDD optimization algorithms not only reduced the number of decision nodes (*node_O*=17838) and redundant 1-paths (*path_O*=19804) efficiently, but also showed their time saving (*time_O*=1505 ms) feature. The TRS approach accurately represented the requirements of the customer, therefore all its 1-paths were valid paths. By setting *path_TRS* as a benchmark, the differences between *path_TRS* and the 1-path number of other approaches were semantically redundant 1-paths. Thus, the *time_I* and *time_O* were greater than the *time_TRS*, since they had to use an extra algorithm to verify the invalid 1-paths during the SLA negotiation. Experimentally, it was proved that after running 1-path verification, the rest 1-paths of the initial ROBDD and the ROBDD by applying path and node optimization was exactly the same as the one of the TRS ROBDD with respect to quantity and semantic meaning.

In summary, the TRS/node optimization led to the most reduction in number of 1-paths (65.43%) and negotiation time (27.75%), although the number of decision nodes had an increase of 19.43%. Furthermore, BDD node/path optimizations led to the most reduction in number of decision nodes, which was 19.37%. Moreover, they had reduction in number of 1-paths (58.64%) and negotiation time (24.07%), but it may still not completely eliminate all the semantically redundant 1-paths, for which reason, the SLA manager requires more time to verify 1-paths during the negotiation. However, it could be a good eclectic approach. From another standpoint, for example, at 1000 ms of the SLA negotiation, the respective SLA number of three approaches were $SLAs\_I$=397, $SLAs\_TRS$=650 and $SLAs\_O$=605. Therefore, the TRS approach had the most SLAs and potentially led to more profits and higher customer satisfaction.

6 Conclusions and Future Work

The negotiation of SLAs in service computing overall, and cloud computing in specific, can be supported by the modeling of SLAs as ROBDDs. Nevertheless, ROBDDs may be suboptimal in structure. In this paper, we show that by applying the BDD node optimization, the number of nodes can be decreased efficiently. Thus, SLAs occupy less memory space. Additionally, the size of semantically redundant 1-paths can be eliminated through the BDD path optimization. Consequently, the SLA negotiation is accelerated by assessing fewer SLA proposals (1-paths). Furthermore, we build on the observation that the options of an SLA term may be mutually exclusive. Thus, an SLA term is rewritten as correctly selecting an option for this term. This process can be treated as an implementation of an *SLA term rewriting system*. Hence, the approach above can eliminate the number of semantically redundant 1-paths at BDD creation phase. Finally, we discuss the strengths and weaknesses of the approaches with an IaaS use case.

In the future, we wish to apply TRS to SLA translation. Furthermore, a suitable representation and transformation would need to be defined to be able to use term rewriting into our scenario. Besides, there is a gap in using the canonical form of the structure for outsourcing and decision-making, related to matching paths from different BDDs and finding out whether they are equivalent so that the outsourced requirements match the available services from sub-contractors.

References

1. Vázquez, T., Huedo, E., Montero, R.S., Llorente, I.M.: Evaluation of a Utility Computing Model Based on the Federation of Grid Infrastructures. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 372–381. Springer, Heidelberg (2007)
2. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, 7–18 (2010)
3. Lu, K., Roeblitz, T., Chronz, P., Kotsokalis, C.: SLA-Based Planning for Multi-Domain Infrastructure as a Service. In: 1st International Conference on Cloud Computing and Services Science, pp. 343–351. Springer (2011)
4. Lu, K., Roeblitz, T., Yahyapour, R., Yaqub, E., Kotsokalis, C.: QoS-aware SLA-based Advanced Reservation of Infrastructure as a Service. In: Third IEEE International Conference on Coud Computing Technology and Science (CloudCom 2011), pp. 288–295. IEEE Computer Society (2011)
5. Antonopoulos, N., Gillam, L.: Cloud Computing: Principles, Systems and Applications. Springer (2010)
6. Shannon, C.E.: A symbolic analysis of relay and switching circuits. AIEE (57), 713–723 (1938)
7. Wu, L.L., Buyya, R.: Service Level Agreement (SLA) in Utility Computing Systems. Architecture, 27 (2010)
8. Chazalet, A.: Service Level Checking in the Cloud Computing Context. In: IEEE 3rd International Conference on Cloud Computing, pp. 297–304 (2010)
9. Ron, S., Aliko, P.: Service level agreements. Internet NG project (2001)
10. Ebendt, R., Drechsler, R.: Advanced BDD Optimization. Springer (2005)

11. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: IEEE/ACM International Conference on Computer-Aided Design, pp. 8–15. IEEE Computer Society Press, Los Alamitos (1993)
12. Andersen, H.R.: An Introduction to Binary Decision Diagrams, pp. 8–15. Citeseer (1999)
13. JavaBDD (2007), http://javabdd.sourceforge.net/
14. Klop, J.W.: Term Rewriting Systems. Stichting Mathematisch Centrum, Amsterdam (1990)
15. Baader, F., Nipkow, T.: Term Rewriting and All That, pp. 1–2, 34–35. Cambridge University Press (1999)
16. Open Grid: Web Services Agreement Specification (2007), http://www.ogf.org/
17. Keller, A., Ludwig, H.: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management, 57–81 (2003)
18. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1 W3C Note, World Wide Web Consortium (2001)
19. Kearney, K.T., Torelli, F., Kotsokalis, C.: SLA*: An abstract syntax for Service Level Agreements. In: GRID, pp. 217–224 (2010)
20. Kotsokalis, C., Yahyapour, R., Rojas Gonzalez, M.A.: Modeling Service Level Agreements with Binary Decision Diagrams. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 190–204. Springer, Heidelberg (2009)
21. Amazon EC2 Cloud (2012), http://aws.amazon.com/ec2/
22. Bartlett, J.: Best Practice for Service Delivery. The Stationery Office (2007)
23. Chandra, A., Goyal, P., Shenoy, P.: Quantifying the benefits of resource multiplexing in on-demand data centers. In: 1st ACM Workshop on Algorithms and Architectures for Self-Managing Systems (2003)
24. Machado, G.S., Stillerm, B.: Investigations of an SLA Support System for Cloud Computing. In: Praxis der Informationsverarbeitung und Kommunikation (2011)
25. Open Cloud Computing Interface Specification (2012), http://occi-wg.org/about/specification/
26. Opennebula (2012), http://opennebula.org/
27. 74LVC1G386, 3-input Exclusive-Or gate, Data Sheet, NXP B.V. (2007)
28. Prasad, P.W.C., Raseen, M., Senanayake, S.M.N.A., Assi, A.: BDD Path Length Minimization Based on Initial Variable Ordering. Journal of Computer Science (2005)
29. SLA@SOI (2011), http://sla-at-soi.eu/
30. Yaqub, E., Wieder, P., Kotsokalis, C., Mazza, V., Pasquale, L., Rueda, J., Gomez, S., Chimeno, A.: A Generic Platform for Conducting SLA Negotiations. In: Wieder, P., Butler, J., Yahyapour, R. (eds.) Service Level Agreements For Cloud Computing, Part 4, pp. 187–206. Springer (2011)
31. Drechsler, R., Guenther, W., Somenzi, F.: Using lower bounds during dynamic BDD minimization. IEEE Trans. on CAD, 50–57 (2001)
32. Ehrgott, M.: Multicriteria Optimization, 2nd edn., pp. 171–195. Springer (2005)