Efficient Model Checking of IT Change Operations

Sebastian Hagen, *Member, IEEE*, Weverton Luis da Costa Cordeiro, *Member, IEEE*, Luciano Paschoal Gaspary, *Member, IEEE*, Lisandro Zambenedetti Granville, *Member, IEEE*, and Alfons Kemper

***Abstract*—The success of businesses in modern organizations heavily depends on the high availability of information technology (IT) infrastructures. To prevent business disruption, IT operators have worked hard to ensure that any changes to this infrastructure are properly and efficiently deployed. Change management—a discipline of the Information Technology Infrastructure Library (ITIL)—provides important guidance to help achieve this end. As IT infrastructures grow larger, however, ensuring that changes are harmless to business continuity becomes increasingly complex. In fact, previous research has shown that existing approaches for verifying changes suffer from severe scalability issues. This problem can become a serious threat to most organizations, as it can lead for example to customer dissatisfaction due to missed deadlines in service change deployment. To bridge this gap, we propose a partial-order reduction model checking paradigm and algorithm for efficiently detecting harmful change operations. Our model improves the complexity of verifying a set of concurrent change activities against safety constraints by reducing—without losing effectiveness—the verification scope. To prove concept and technical feasibility, we carried out an extensive performance evaluation of our algorithm considering a variety of change activities, safety constraints, and configuration scenarios. The results obtained from 32 benchmarks have shown that our algorithm significantly outperformed state-of-the-art, general purpose model checkers, improving the runtime complexity from polynomial/exponential to linear. In summary, the results evidenced that change verification finally became feasible and efficient for larger IT infrastructures.**

***Index Terms*—Change management, verification, model checker, partial order reduction.**

I. Introduction

DEALING effective and efficiently with changes on information technology (IT) infrastructures is imperative for the success of modern organizations, given that these organizations are increasingly relying on IT to deliver their services to customers. For example, BMW Financial Services Germany faces roughly 3600[1] IT change operations every year; those changes need to be deployed over software and hardware systems that provide financial services 24 × 7. This number is comparatively small, however, if one consider those changes occurring in cloud providers such as Microsoft and Amazon; in their data centers, the volume of changes can easily exceed the order of hundreds per day. These changes may vary in nature and complexity, for example change the configuration of a router, deploy a new application, or migrate an entire virtual network infrastructure to a different facility. In this context, IT operators have been working hard to ensure that these changes are properly deployed, since failures can lead for example to business disruption and thus customer dissatisfaction. In fact, severe outages can occur because of faulty deployment of change operations. In April 2011, an inadequate network change at one of Amazon's data centers caused a one-week outage to customer services [1]. To prevent such failures, change management [2]—a discipline of the Information Technology Infrastructure Library (ITIL) [3]—describes a generic process for properly handling changes, from the design and planning phases to the evaluation, authorization, testing, scheduling, deployment, and review ones.

In practice, the success of a change management process heavily depends on the experience of IT operators, being rarely supported by automated tools and algorithms. The use of algorithms to support change management has potential to increase the reliability and efficiency of the process, and make it less dependent on the individual skills of IT personnel. In particular, the evaluation, authorization, and testing phases are well suited to be supported by the automated verification of IT change operations. Verification can prove, on a logical level, that a set of change operations will not invalidate hosting safety constraints. Thus, assuming that a logical model of the IT environment correctly captures reality, a verification process over that model can early detect the occurrence of threatening conditions—for example, a network overload as in Amazon's case—automatically and efficiently.

While the verification community has extensively discussed verification algorithms [4], [5], previous work applying them to IT changes [6], [7], suffer from a common scalability drawback. To illustrate, a report from Data Center Knowledge[2] estimates that Amazon Cloud operates about 450 000 servers; Google, about 900 000 servers; and Microsoft, over than 1 000 000 servers. Note that the estimates do not consider the multitude of devices (switches, routers, etc.) that provide connectivity between them, software installed on the servers, and their intricate set of relationships. In the case of cloud providers, the mapping between virtual entities and physical ones poses an extra challenge to the operators responsible for these infrastructures. In

Manuscript received June 28, 2013; revised December 9, 2013 and May 19, 2014; accepted May 21, 2014. Date of publication August 7, 2014; date of current version September 5, 2014. The associate editor coordinating the review of this paper and approving it for publication was C. Bartolini.

S. Hagen and A. Kemper are with Technische Universität München, 80333 München, Germany.

W. L. da Costa Cordeiro is with the Federal Institute of Para, 68180-000 Itaituba-Para, Brazil.

L. P. Gaspary and L. Z. Granville are with the Federal University of Rio Grande do Sul, 91509-900 Porto Alegre-RS, Brazil (e-mail: paschoal@inf.ufrgs.br).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNSM.2014.2346074

[1]This number was obtained from a conversation the authors had with the change management staff working at BMW financial services Germany.

[2]http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-most-web-servers

this context of super-sized and overly complex infrastructures, existing verification algorithms quickly hit a scalability barrier. This barrier makes these algorithms unfeasible to realistically sized verification problems. To solve this issue, we propose an Extended Partial-Order Reduction (EPOR) model checking paradigm that reduces—without losing effectiveness—the verification scope for a set of concurrent change activities against safety constraints. Compared to general purpose model checkers, our model significantly decreases change verification runtime complexity from polynomial/exponential to linear.

We evaluated our algorithm using 32 benchmarks, which comprised a variety of model checker specific optimization techniques, network change operations, and infrastructure configurations that have caused the outage in one of Amazon's data centers. In summary, our evaluation has shown that our algorithm outperformed the state-of-the-art, general purpose model checkers SPiN [8], [9] and NuSMV [10]. It has also shown that our algorithm is more robust against bad modeling choices in the specification of changes. In other words, operators inexperienced in the formalization of efficiently verifiable models are less penalized (in terms of verification complexity) by such bad choices; even the worst-case ones for our algorithm easily outperform the best performance achievable by the SPiN and NuSMV model checkers. Thus, to the best of our knowledge, this is the first paper to propose a solution that makes change verification feasible on large scale infrastructures while providing robustness for untrained staff.

The remainder of this paper is organized as follows. Section II discusses related work and contributions. The theory of Extended Partial-Order Reduction (EPOR) is introduced in Section III, followed by a logic for change verification in Section IV. Section V introduces the Amazon network outage, the change activities and safety constraints considered, and their description in change verification logic. The experimental setup is described in Section VI, followed by the evaluation in Section VII. Finally, Section VIII concludes the paper.

II. Related Work and Contributions

Our preliminary work on IT change verification [11] introduces a factorial algorithm to this end, which does not scale for very large infrastructures. A subsequent work [12] presents the preliminary ideas of extended partial-order reduction. However, that work lacks the formal theory established herein and the evaluation (32 benchmarks) of extended partial-order reduction against the NuSMV and SPiN model checkers.

The work by Kikuchi *et al.* [6] and Radu *et al.* [7] are those most related to our research. In these papers, the authors investigate the use of model checkers to detect policy conflicts in autonomic computing systems. Using the SPiN model checker, the authors conclude—similarly to us—that problem instances need to be small enough to remain feasible for verification. However, extended partial-order reduction extends their results by significantly increasing the scalability of change verification to large configurations. Calinescu *et al.* [13] discuss several challenges to apply formal methods in the context of distributed systems and configurations. Our work complements that one by solving the scalability challenge.

The detection of conflicts, particularly in the field of network security, has attracted significant interest. Al-Shaer *et al.* [14] propose an algorithm for the automated detection of conflicts in firewall policies. The notion of conflicts is restricted to specific anomalies in the configuration of these policies. Different to our work, conflict detection can be performed over a static set of policies and does not need to take into account the effects of change activities on the configuration. Hu *et al.* [15] expresses access control models and properties for verification using the NuSMV model checker. Consequently, their work underlies the same scalability issues as observed in our evaluation. Uszok *et al.* [16] implement an algorithm to detect policy conflicts that incorporate authorizations and obligations. Common to all these works is that they are not expressive enough to be used for change verification as they have been engineered for very specific cases.

There have also been relevant research efforts to improve other aspects of IT service management. For example, Bartolini *et al.* [17] address the simulation and optimization of the IT incident management process to improve the handling of tickets in a service organization. Sauvé *et al.* [18] propose the prioritization of changes based on the assessed risks. Our work profits from risk assessment because it provides important clues about critical change activities worthy to be verified. Another active research area trying to make the execution of changes more reliable has been that of change planning. The authors of [19]–[21], for example, proposed algorithms for the automated planning of IT changes. Similar to verification, planning aims to make change management more reliable by means of automation, but cannot guarantee the correctness of changes happening beyond the planning system.

In the area of verification, our work is related to Partial Order Reduction (see Baier *et al.* [4], chapter "Partial Order Reduction"). Different to POR, our proposal (Extended Partial Order Reduction) is only applicable to more restricted formulas, but remains computationally feasible in cases for which partial-order reduction cannot be applied.

III. Extended Partial-Order Reduction

A. Basic Definitions: IT Change Verification Problem

This section establishes basic notation used throughout this work. $S = \{s_1, \ldots, s_n\}$ denotes a set of states; each state describes the configuration of a Configuration Management Database (CMDB), i.e., the relationships between software and hardware systems in a data center. We herein use the notion of state and configuration interchangeably. ϕ denotes a formula that may or may not hold in a state $s \in S$. For a formula ϕ and a state $s \in S$ we write $s \models \phi$ if ϕ holds in s and $s \not\models \phi$ otherwise. $E = \{e_1, \ldots, e_n\}$ denotes a set of effects where $e \in E$, $e : S \to S$ is a function that is applied to a state $s \in S$ and yields a successor state $e(s) \in S$ that describes the state that evolves from s by the application of effect e.

Next we define our understanding of IT change activities and safety constraints. A change activity $act \in \{act_1, \ldots, act_n\}$ is a subset of effects, i.e., $act \subseteq E$, that describes the modifications an IT change has on $s \in S$. A safety constraint is a

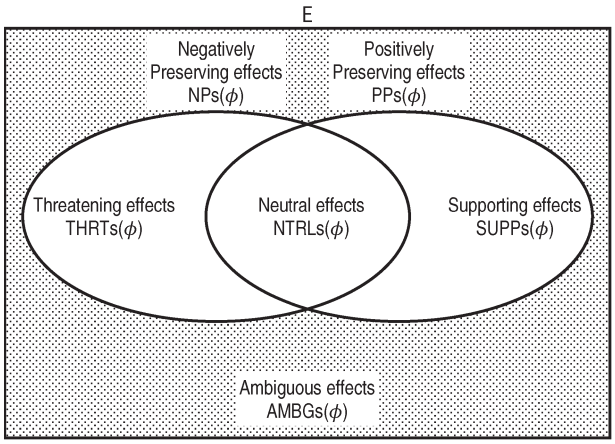

Fig. 1. Relationships between effects and a formula ϕ.

formula ϕ that either holds (write $s \models \phi$) in $s \in S$ or not. To reason about how effects modify the state of a CMDB, we consider sequences of effects $\langle e_1, \ldots, e_n \rangle$, $n \in \mathbb{N}_0$, where $\forall i \in \{1, \ldots, n\} : e_i \in E$. The application of sequences of effects to states is defined as the subsequent application of effects. Thus for a state s and a sequence of effects $\langle e_1, \ldots, e_n \rangle$ we obtain $\langle e_1, \ldots, e_n \rangle(s) = e_n(\ldots(e_1(s)))$. We denote by $Seq(E)$ the set of all effect sequences of arbitrary length (between 0 and $|E|$) such that every effect in E occurs at most once in every sequence $seq \in Seq(E)$. Then, for a set of safety constraints $\{\phi_1, \ldots, \phi_n\}$ the IT change verification is the problem to determine whether

$$\forall seq \in Seq(E) : \quad seq(s) \models \phi_i$$

holds for every safety constraints $\phi_i \in \{\phi_1, \ldots, \phi_n\}$, i.e., to decide whether all safety constraints hold under any possible execution sequence of effects of change activities. Deciding this problem naively means to consider $O(|E|!)$ arrangements.

In our first work [11] on IT change verification, we proposed to consider an approach like that. Its scalability however is very limited. State of the art model checkers, such as the SPiN and NuSMV, can solve the IT change verification problem in exponential or polynomial time. However, even polynomial runtime complexity does not scale to realistic CMDB sizes. Next we show that, with some additional requirements, it suffices to consider only a single sequence of effects in E.

B. Relationships Among Effects and Formulas

Fig. 1 depicts the partition of all effects E in support, threats, neutrals, and ambiguous effects. For a formula ϕ, every effect in E belongs to exactly one of these classes. The following definitions describe in more detail each of them.

Definition 1: Let $e \in E$ be an effect. We then have that e is a "positively preserving" effect in respect to a formula ϕ iff

$$\forall s \in S : (s \models \phi \rightarrow e(s) \models \phi)$$

We denote by $PPs(\phi) \subseteq E$ all effects in E that positively preserve formula ϕ.

Definition 2: Let $e \in E$ be an effect. We then have that e is a "negatively preserving" effect in respect to a formula ϕ iff

$$\forall s \in S : (s \not\models \phi \rightarrow e(s) \not\models \phi)$$

We denote by $NPs(\phi) \subseteq E$ all effects in E that negatively preserve formula ϕ.

Positively preserving effects are good ones, as they always preserve the *true* value of a formula. They never turn a formula *false* if it already holds. Negatively preserving effects are bad ones, in the sense that they keep a formula *false* once it does not hold. They never can turn a formula from *false* to *true*. To illustrate, consider a CMDB state s_1 in which some computer c has 8 GB of physical memory installed, and a formula ϕ which states that c must have at least 8 GB of physical memory. An example of a positively preserving effect in this context is an activity that adds 2 GB of physical memory to c. Conversely, a negatively preserving effect is an activity that decreases in 2 GB the amount of memory available in c.

Based on positively and negatively preserving effects, we further categorize effects as supporting (Def. 3), threatening (Def. 4), neutral (Def. 5), and ambiguous (Def. 6) ones. Notice that an effect $e \in E$ can appear in different roles in respect to different formulas.

Definition 3: Let $e \in E$ be an effect and ϕ a formula. Then

$$SUPPs(\phi) = PPs(\phi) \cap \overline{NPs(\phi)} \subseteq E$$

are called supports supporting effects.

Def. 3 involves those effects which preserve the *true* value of a formula, and may turn it from *false* to *true* on some state s. For a practical example, consider a CMDB state s in which some computer c has 4 GB of memory, and a formula ϕ which states that c must have at least 6 GB of memory. In this context, an activity that adds 1 GB of memory to c has a supporting effect over formula ϕ, since it can turn ϕ *true* if executed twice, and keep it *true* if executed repeatedly.

Definition 4: Let $e \in E$ be an effect and ϕ a formula. Then

$$THRTs(\phi) = \overline{PPs(\phi)} \cap NPs(\phi) \subseteq E$$

are called threats threatening effects.

Def. 4 involves those effects which preserve the *false* value of a formula, and may turn it from *true* to *false* on some state s. To illustrate, consider the previous CMDB state and formula ϕ. An activity that decreases 1 GB of memory to c has a threatening effect over formula ϕ, since it can turn ϕ *false* if executed twice, and keep it *false* if executed repeatedly.

Definition 5: Let $e \in E$ be an effect and ϕ a formula. Then

$$NTRLs(\phi) = PPs(\phi) \cap NPs(\phi) \subseteq E$$

are called neutrals neutral effects.

Def. 5 involves those effects which are positively and negatively preserving effects at the same time. They are easy to reason about because they always preserve the value of a formula when applied to any state. Focusing on the practical example described earlier, a change activity that modifies the clock speed of the physical memory installed in c has a neutral effect over ϕ, since the total amount of physical memory available is not changed.

Definition 6: Let $e \in E$ be an effect and ϕ a formula. Then

$$AMBGs(\phi) = \overline{PPs(\phi) \cup NPs(\phi)} \subseteq E$$

are called ambiguous effects.

Def. 6 (ambiguous effects) involves those effects which are not supporting, threatening, or neutral effects. These effects are difficult to reason about as they show different behavior depending on a state. For an ambiguous effect of a formula ϕ there exists a state where the application of the effect turns ϕ from *true* to *false*, but there also exists a state for which it behaves the other way round. To illustrate, suppose a configuration item x, a CMDB state in which $x = 5$, and a formula $\phi\ x \geq 5$. A change activity *multiply* $(x, -1)$ is ambiguous, since it may turn formula ϕ true or false depending on the current value of x.

Until now, we have not formulated any requirements for effects and formulas to be efficiently verifiable. This is done by the definitions described next.

Definition 7: Let $supp \in SUPPs(\phi)$ be a support. $supp$ is called threat-independent support iff

$$\forall ts \in Seq\left(THRTs(\phi)\right), \quad \forall s \in S : \\ ts(s) \models \phi \quad \rightarrow \quad ts\left(supp(s)\right) \models \phi \quad (1)$$

Definition 8: Let $thr \in THRTs(\phi)$ be a threat. thr is called threat-independent iff

$$\forall ts \in Seq\left(THRTs(\phi) \setminus \{thr\}\right), \quad \forall s \in S : \\ ts(s) \not\models \phi \rightarrow ts\left(thr(s)\right) \not\models \phi \quad (2)$$

The occurrence of threat-independent supports and threats is an important requirement to efficiently solve the IT change verification problem (see Section III-A). Def. 7 (threat-independent supports) involves those support effects that can be inserted before a sequence of threats, while preserving the evaluation of ϕ in one direction. For a support to be also threat-independent, the original definition of a positively preserving effect ($\forall s \in S : s \models \phi \rightarrow supp(s) \models \phi$) (see Def. 1) still has to hold after an arbitrary sequence of threats has been applied to $s \in S$ on both sides of the implication, s and $supp(s)$.

Def. 8 (threat-independent threats) is analogous to the previous definition: it involves those threat effects that can be inserted at the beginning of a sequence of threats to a formula while preserving the evaluation of ϕ in one direction.

The definitions of threat-independent supports and threats are necessary to drastically reduce the number of effect sequences that need to be checked to verify a formula. If we also assume that threats are permutable, i.e., permuting threats in a sequence of threats does not change the evaluation of a formula, it can be shown that threat-independent supports and threats are negligible for IT change verification:

Theorem 1: Let ϕ be a formula, $s \in S$ the configuration of a CMDB, and ts an arbitrary sequence of threats, such that $ts \in Seq(THRTs(\phi))$ and $|ts| = |THRTs(\phi)|$. Furthermore, let (i) all supports and threats of ϕ be threat-independent, and (ii) all threats of ϕ be permutable. Then,

$$ts(s) \models \phi \quad \leftrightarrow \\ \forall seq \in Seq(THRTs(\phi) \cup SUPPs(\phi)) : seq(s) \models \phi$$

Theorem 1 is quite powerful[3]: it tells us that we only need to consider an arbitrarily ordered sequence of all threat-independent threats to decide whether a formula holds under any possible interleaved execution sequences of threat-independent threats and supports. While the left side of the equivalence can be evaluated in time linear in the number of threats, the right side is factorial in the number of threat-independent threats and supports. Broadly speaking, the idea underlying Theorem 1 is that, among threat-independent supports and threats, only the threats need to be considered. This is because they are the only bad effects that can invalidate a formula. For now, we have shown that in sequences comprising threat-independent supports and threats, only threats need to be considered in one order (assuming the requirements of Theorem 1). But how do we deal with ambiguous and neutral effects? Ambiguous effects cannot be dealt with efficiently. They can turn a formula in both directions, depending on the state. We explicitly need to exclude their occurrence to efficiently solve the IT change verification problem. To properly deal with neutral effects, we require a similar, but slightly stronger criterion than threat-independence, to hold for neutral effects as well (effect-independent neutral).

Definition 9: Let ϕ be a formula and $ntr \in NTRLs(\phi)$ a neutral effect. ntr is called effect-independent iff

$$\forall s \in S, \quad \forall seq \in Seq\left(E \setminus \{ntr\}\right) : \\ seq(s) \models \phi \quad \leftrightarrow \quad seq\left(ntr(s)\right) \models \phi \quad (3)$$

Effect-independent neutrals can be added and removed to/from the beginning of any sequence of effects while the evaluation of a formula is not changed. This means that all effect-independent neutrals can be gradually removed from any sequence of effects without changing the evaluation of a formula after the application of the effect sequence. Thus, if the requirements of Theorem 1 hold and there are no ambiguous effects, we can combine this observation with Theorem 1 yielding an efficient theorem to verify a formula:

Theorem 2: Let ϕ be a formula, $s \in S$ the configuration of a CMDB, and ts an arbitrary sequence of threats, such that $ts \in Seq(THRTs(\phi))$ and $|ts| = |THRTs(\phi)|$. Furthermore, let (i) all requirements of Theorem 1 be satisfied, (ii) all neutrals of ϕ be effect-independent, and (iii) $AMBGs(\phi) = \emptyset$. Then,

$$(\forall seq \in Seq(E) : \quad seq(s) \models \phi) \qquad \leftrightarrow \qquad ts(s) \models \phi$$

Proof: Because $E = SUPPs(\phi) \cup THRTs(\phi) \cup NTRLs(\phi) \cup AMBGs(\phi)$ (see Fig. 1) and $AMBGs(\phi) = \emptyset$, the left side of the equivalence is equivalent to

$$\forall seq \in Seq\left(SUPPs(\phi) \cup THRTs(\phi) \cup NTRLs(\phi)\right) : \\ seq(s) \models \phi$$

By applying Def. 9 to each effect-independent neutral in every sequence seq we obtain logical equivalence with

$$\forall seq \in Seq\left(SUPPs(\phi) \cup THRTs(\phi)\right) : \quad seq(s) \models \phi$$

and with Theorem 1 we finally obtain equivalence with $ts(s) \models \phi$. ■

[3]We refer the interested reader to a technical report from our group [22] for a complete proof of Theorems 1 and 2.

TABLE I
SUPPORT, THREATS, AND NEUTRAL RELATIONSHIPS OF THE EFFECTS AND PREDICATES OF THE CHANGE VERIFICATION LOGIC. (a) ARITHMETIC EFFECTS AND PREDICATES. (b) SETTING AND CHECKING OF PROPERTIES. (c) EFFECTS AND PREDICATES ON SETS AND LISTS

	Predicates	
Effects	$y \geq / > x$	$y \leq / < x$
$inc(x)$	threat	support
$inc(y)$	support	threat
$dec(x)$	support	threat
$dec(y)$	threat	support

(a)

	Predicates	
Effects	$hasValue(x)$	$!hasValue(x)$
$set(x)$	support	threat
$set(y)$	threat	support

(b)

	Predicates	
Effects	$contains(x)$	$!contains(x)$
$add(x)$	support	threat
$add(y)$	neutral	neutral
$remove(x)$	threat	support
$remove(y)$	neutral	neutral

(c)

Theorem 2 describes the requirements that a change verification logic needs to adhere to be efficiently verifiable. If threats to a formula can be determined in constant time, Theorem 2 provides us with a criterion linear in the number of threats of a formula ϕ to determine whether ϕ holds under any sequence of effects. However, deciding the relationships among effects and formulas can become complex for compound formulas. Luckily, it can be shown that, for a compound formula $\phi_1 \wedge \phi_2$, Theorem 2 can be recursively applied to ϕ_1 and ϕ_2 to decide whether $\phi_1 \wedge \phi_2$ holds under any possible execution sequence of effects. The same holds (with some restrictions) for formulas composed using $\vee$. Thus, we only need to determine threats, supports, neutrals, and ambiguous effects at the level of atomic propositions, i.e., predicates, of a change verification logic and then evaluate the compound formula bottom up. This also means that it suffices to prove the theorems and definitions given on the level of effects and atomic propositions, and not for compound formulas.

IV. CHANGE VERIFICATION LOGIC AND ALGORITHM

In the previous section, we described the set of definitions and theorems that enables us to significantly reduce the complexity of evaluating the validity of a set of effects against a current state of an IT infrastructure. In this section, we described how the theory presented earlier can be combined with the theory of Extended Partial Order Reduction (EPOR) for the purposes of efficient IT change verification. We begin by introducing an EPOR-compliant logic to describe change activities and safety constraints. Then, we present our algorithm for efficient verification of IT changes.

A. Change Verification Logic

To exploit EPOR for IT change verification, a logic that is expressive enough to describe a wide range of change activities and safety constraints but adheres to the theorems of EPOR is required. While one can be easily overwhelmed by the many change activities and safety constraints such a logic should be able to describe, a step backwards needs to be made to determine an expressive set of effects that integrates with the way how configurations of data centers are described.

In commercial Configuration Management Database (CMDB) systems [23] and modeling standards such as the Common Information Model (CIM) [24], configurations are described as an object-oriented (OO) graph, i.e., an OO instance diagram. Such a model/graph comprises the configuration items, their properties, and references among them. As commercial CMDB systems automatically collect and update the OO instance graph, we address in this work those change activities whose effects are reflected by modifications to an OO instance graph. Thus, covering effects that describe a wide range of changes to OO-instance graphs yields a logic expressive enough to describe the effects of change activities within the limits of what current state of the art CMDB products are able to record and detect today.

The modifications on OO instance graphs largely depend on the modification operations supported by the data types of properties of the model. Typical data types of properties of configuration items in CIM [24] models are arithmetic data types (e.g., Integer, short, real, etc.), references (including aggregation and composition), strings, lists, and sets. Table I provides an overview of the effects and predicates, i.e., atomic propositions, out of which formulas in change-verification logic are formed using the standard logical connectors $\wedge$,$\vee$,$\rightarrow$,$\leftrightarrow$. The change verification language supports effects to add $(add(l,x))$ and remove $(remove(l,x))$ an element x from a list or set l and predicates to check whether x is comprised in l $(contains(l,x))$ or not $(!contains(l,x))$. These effects are useful when to describe change activities that affect 1:N or N:M relationships among configuration items, e.g., when a virtual machine is added to a physical machine, or to describe a list of destination ports assigned a specific mark for routing purposes (for instance see the *shift traffic* change activity in Section V-C).

Furthermore, effects to set the value of a property of a configuration item from p to $x(set(p,x))$, and to check whether it has that value $(hasValue(p,x))$, are supported. This is useful when to set and check references among configuration items, e.g., when the default gateway of a routing table needs to be changed (for instance in the *failover* change activity in Section V-C). Furthermore, the change verification language supports effects to increase $(inc(x,\Delta c))$ or decrease $(dec(x,\Delta c))$ an arithmetic property x by a constant $\Delta c > 0$. The predicates supported on arithmetic properties are the standard arithmetic binary comparison operators $(>,\geq,<,\leq)$. Arithmetic effects and predicates are useful to describe changes and constraints regarding the metrics of network interfaces, e.g., safety constraint SC2 and change activities $HCRIncrM/LCRDecrM$ in Section V-D.

The effects and predicates of the change verification language (see Table I) satisfy all theorems of EPOR. Proving this is a process that takes the semantics of effects and predicates into account; We refer the interested reader to a technical report from our group [22] for a detailed, comprehensive demonstration of this aspect.

B. Algorithm

The EPOR verification algorithm is a straight-forward implementation of Theorem 2. As the pseudo code has been previously provided in [12], we only give a textual description herein. The algorithm comprises three steps. First, the algorithm determines at the level of effects of change activities and predicates comprised in the formulas of safety constraints the relationships according to Table I. Second, it uses Theorem 2 to determine whether each predicate comprised in a safety constraints holds under any interleaved execution sequence of effects. To do that, each predicate needs to be evaluated over a state that has seen the application of all the predicate's threats to the initial state (see Theorem 2 instantiated at the level of predicates). Third, once the truth values of all predicates have been obtained on those states the (usually) more complex compound formulas of safety constraints can be evaluated bottom up using the standard interpretation of $\wedge$,$\vee$,$\rightarrow$, and $\leftrightarrow$. If the formula of a safety constraints then holds, so does it on any configuration evolving from the initial state by the interleaved execution of effects of change activities.

V. Case Study: Amazon Network Outage

To assess the efficacy and effectiveness of our solution for efficient verification of IT changes, we considered in our evaluation an simplified instance of the Amazon's *Elastic Block Store* (EBS) infrastructure, and focus on the change request that caused a network overload on it. In this section, we briefly describe the portion of the IT infrastructure involved in the change, the causes of the network overload, and the set of change activities (later considered in our experimental evaluation) that contributed to the overload.

A. The Elastic Block Store

Amazon's *Elastic Block Store* (EBS) is a "*distributed, replicated block data store [...] [for] read and write access from EC2 instances*" [1], Amazon's virtual machines. The EBS system consists of several clusters each comprising several EBS nodes/servers that "*store replicas of EBS volume data and serve read and write requests to EC2 instances*" [1]. Amazon's incident report [1] mentions two types of networks: (1) a *primary network*, which is a "*high bandwidth network used in normal operation for all necessary communication with other EBS nodes, with EC2 instances, and with the EBS control plane services*" [1], and (2) a *secondary network*, also called the replication network, which is a "*lower capacity network used as back-up to allow EBS nodes to reliably communicate with other nodes in the EBS cluster and provide overflow capacity for data replication*" [1]. Thus, a routing policy is needed to diverge the outgoing traffic to the different gateways of the high- and low-capacity network because the low-capacity one is not designed to "*handle all traffic from the primary one but rather provide[s] highly-reliable connectivity between EBS nodes inside of an EBS cluster*" [1]. The scope of the low-capacity network is limited to the cluster and the high-capacity network provides communication beyond the cluster to EC2 instances and cloud interfaces.

B. Network Overload Caused by Traffic Shift

The outage in April 2011 was triggered by a *Request for Change* [2] "*to upgrade the capacity of the primary network*" [1]. The request was further refined into a set of IT change activities, among them one to "*shift the traffic off of one of the redundant routers in the primary EBS network to allow the upgrade to happen*" [1]. Next, a router had to be chosen to re-reroute the traffic to. "*The traffic shift was executed incorrectly and rather than routing the traffic to the other router on the primary network, the traffic was routed onto the lower capacity redundant EBS network*" [1]. Thus, we draw the following conclusions: (1) every EBS server node needs to have two gateways of the high-capacity network on its virtual/physical subnet it can route replication traffic to, and (2) it needs to have a gateway of the low-capacity network on the same subnet to route its reliable traffic to.

As a consequence of the incorrect traffic shift, "*a portion of the EBS cluster did not have a functioning primary or secondary network, because traffic was purposely shifted away from the primary network and the secondary network could not handle the traffic level it was receiving*" [1]. The result was that "*many EBS nodes [...] were completely isolated from other EBS nodes*" [1]. The primary routers of the high-capacity network were unavailable for inter-subnet communication within an EBS cluster and the low-capacity network was overloaded. The communication outage finally triggered a long chain of destabilizing events which led to unavailability of services and data loss.

The incident report does not state how the traffic shift was achieved. In the following sections, we hypothesize on methods to achieve such a traffic shift, based on best practices in data center network design [25].

C. Network Overload in Static Routing Environment

In static routing environments servers of the EBS cluster are configured with a static gateway IP address of a router interface. To achieve routing redundancy, the virtual IP address of the gateway is shared among a group of routers that run a *first hop redundancy protocol*. Typical first hop redundancy protocols are, for example, Cisco's proprietary *Hot Standby Router Protocol*, the *Gateway Load Balancing Protocol*, or the IETF standard *Virtual Router Redundancy Protocol*. A first hop router redundancy protocol aggregates interfaces from different routers to a group that shares a virtual IP and virtual MAC address. The servers route outgoing traffic to the redundant virtual IP address and Layer 2 switches forward frames to the physical interface that hosts the virtual IP. The interface of a backup router takes over virtual IP and virtual MAC of the master router interface in case of failure and floods the network with the corresponding *Address Resolution Protocol* (ARP) packages via its interface to transparently achieve the network shift on Layer 2.

Four different change activities could have caused Amazon's network overload in this configuration:

- **Failover negative change activity (FOn)**: a FOn change activity occurs when an interface of a high-capacity router is switched off and the configured failover interface

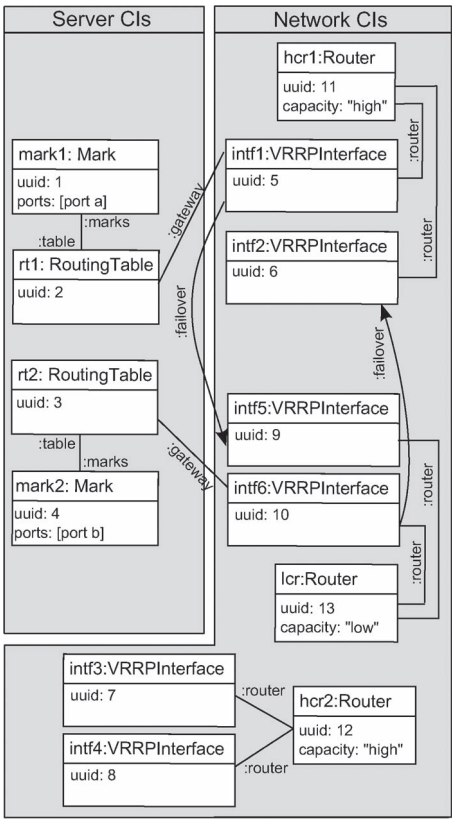

Fig. 2. Example configuration of the CMDB to describe change activities and safety constraints.

belongs to a low-capacity router. This causes a network overload because all nodes of the affected subnet automatically start routing towards the low-capacity interface.

- **Failover positive change activity (FOp)**: a FOp change activity occurs when the failover interface belongs to a high-capacity router. In this case no network overload occurs because the high-capacity router can handle the network traffic, see equation at the bottom of the page. The verification logic template of a failover change activity takes three parameters: (1) current, the interface that is deactivated, (2) failover, the interface which is configured as failover, and (3) the routing table (rt) which routes towards the interface to be deactivated (current). Depending on the parameters of the failover change template, FOp or FOn change activities, can be instantiated from the template. For example, *Failover(intf1,intf5,rt1)* (see Fig. 2 for the configuration items) yields an applicable FOn change activity, because high-capacity traffic (port a traffic) routed via intf1 is taken over by intf5, a low-capacity router interface. In turn, *Failover(intf6,intf2,rt2)* is a FOp change because *intf2*, a high-capacity interface, takes over interface *intf6* over which low-capacity traffic (port b) is routed.

  A different way to achieve the network shift is to directly manipulate the routing policy of each EBS node by changing the mark that is carried by traffic destined for a particular port. Changing the mark of a traffic associates the traffic with a different routing table which has a different default gateway, thus causing a traffic shift.
- **Shift traffic negative change activity (SHTn)**: a $SHTn$ change activity changes the routing policy of an EBS node in such a way that high-capacity traffic (port a traffic) is assigned the mark of low-capacity traffic (port b). Consequently, high-capacity traffic is routed using the routing policy of low-capacity traffic, i.e., towards a low-capacity router interface. This causes an overload of the low-capacity router. $SHTn$ change activities need to be instantiated for every server.
- **Shift traffic positive change activity (SHTp)**: a $SHTp$ change activity assigns the mark of high-capacity traffic to low-capacity traffic. Consequently, low-capacity traffic is routed using high-capacity rules, i.e., towards a high-capacity router interface. $SHTp$ change activities do not cause a network overload.

$\mathbf{SHT}(\mathbf{from}: \mathbf{Mark}, \mathbf{to}: \mathbf{Mark}, \mathbf{port}: \mathbf{int})$	
pred_1:	from.ports.contains(port)
pred_2:	to.ports.!contains(port)
pre:	$\text{pred}_1 \wedge \text{pred}_2$
eff_1:	from.ports.remove(port)
eff_2:	to.ports.add(port)

The property *ports* (a set) of *mark configuration items* of the infrastructure model (see Fig. 2) describes the destination ports of traffic that bares the mark. A mark configuration item references the RoutingTable configuration item, which is used to route the marked traffic, and the table references the default gateway interface of the routing table (see Fig. 2). Using this model, a SHT change activity is logically described by moving an entry between the *ports* properties of two Mark configuration items that represent the marks of high- and low-capacity traffic. Note, that the SHT template can either be instantiated to obtain $SHTp$ or $SHTn$ change activities. For example, *SHT(mark1,mark2,port a)* yields a $SHTn$ change activity because high-capacity traffic (*port a*) is shifted to the low-capacity traffic mark and *SHT(mark2,mark1,port b)* yields a $SHTp$ change activity. See Fig. 2 for the instances of the configuration items. What remains to be specified is the safety constraints that is violated (or not) by the application of $SHTn$

$\mathbf{FO}(\mathbf{current}: \mathbf{Interface}, \mathbf{failover}: \mathbf{Interface}, \mathbf{rt}: \mathbf{RoutingTable})$	
pred_1:	rt.gateway.hasValue(current)
pred_2:	current.failover.hasValue(failover)
pre:	pred_1
eff_1:	rt.gateway.setValue(failover)

and FOn ($SHTp$ and FOp) change activities, see equation at the bottom of the page.

SC1 is the safety constraint that protects the CMDB model depicted in Fig. 2 from a network overload. *SC1* guarantees that, if a mark is routed using a routing table ($pred_4$) that routes to an interface ($pred_3$) which belongs ($pred_2$) to a low-capacity router ($pred_1$), then the mark must not hold for high-capacity traffic (port a, $pred_5$). Safety constraints and change activities are directly evaluated over the configuration of a CMDB to determine whether they are violated. For example, instance *SC1(lcr,intf6,rt2,mark2)* of safety constraint *SC1* evaluates true in the configuration depicted in Fig. 2.

D. Network Overload in Dynamic Routing Environment

Instead of static routing, a data center network can be configured with dynamic routing. Then, routes are installed and withdrawn from routing tables of the EBS nodes by a routing protocol depending on current network conditions. When an EBS node learns of different routes via different gateways to the same network, it installs the cheapest route according to a metric into its routing table. Once the cheapest route becomes unavailable, the next cheapest route is installed into the routing table and a traffic shift occurs. In this work we focus on *Open Shortest Path First* (OSPF), which is arguably the most widely used routing protocol for routing within an autonomous system, e.g., a large data center. It is important to note that the change verification logic is expressive enough to describe any routing protocol because it supports the arithmetic operations (increase and decrease) and comparisons used by routing protocols to determine the cheapest route. Using OSPF, metrics can be either configured manually for an interface or the metric of an interface is automatically computed as the quotient of the reference bandwidth bw_{ref} and the bandwidth of the interface bw_{intf}. Thus, $cost_{intf}$ can only be manipulated by changing bw_{ref} because the bandwidth of an interface is fixed.

In the case of dynamic routing and Amazon's network outage, a network overload occurs when a low-capacity router advertises the cheapest route. In this case, all EBS nodes install the route via the low-capacity interface into their routing tables leading to a network overload. Two scenarios exist that turn a low-capacity interface to be the cheapest one: (1) the metric of a low-capacity router interface is accidentally reduced such that it advertises the newest cheapest route, or (2) the metrics of the high-capacity router interfaces are increased such that the low-capacity router interface advertises the newest cheapest route. Both scenarios cause a network shift towards a low-capacity router interface and consequently a network overload. To describe both scenarios in logic, we introduce the following (simple) changes: (a) $HCRIncrM$, which increases the manually configured costs of an interface of a high capacity router, (b) $HCRIncrOSPF$, which does the same as $HCRIncrM$, but in a context where OSPF is used, (c) $LCRDecrM$, which decreases the manually configured costs of an interface of a low capacity router, and (d) $LCRDecrOSPF$, which does the same as $LCRDecrOSPF$, but in a context where OSPF is used. The CMDB model that can be used to describe these changes can be simple: it suffices to model network interface configuration items with an arithmetic property *cost* that describes the value of the routing metric. In case of automatically computed OSPF metrics, properties for the reference bandwidth and the interface bandwidth are necessary as well.

LCRDecrM(**interface**: **Interface**, **int**: **delta**)
eff$_1$: | interface.cost.dec(delta)

A $LCRDecrM$ change activity decreases the manually configured cost of a router interface by a specified delta. The same change using OSPF metrics requires an additional effect that decreases the reference bandwidth:

LCRDecrOSPF(**interface**: **Interface**, **int**: **delta**)
eff$_1$: | interface.refbw.dec(delta)
eff$_2$: | interface.cost.dec(delta/interface.intfbw)

$HCRIncrOSPF$ and $HCRIncrM$ changes are specified analogously using the inc effect. The safety constraint (*SC2*) to protect from an overload is rather simple: the cost of at least one high-capacity router interface metric needs to be cheaper than that of the low-capacity router interface:

SC2(**intf1−4**: **Interface**, **lowcapintf**: **Interface**)
pred$_1$: | intf1.cost $<$ lowcapintf.cost
pred$_2$: | intf2.cost $<$ lowcapintf.cost
pred$_3$: | intf3.cost $<$ lowcapintf.cost
pred$_4$: | intf4.cost $<$ lowcapintf.cost
pre: | $V_{i=1\ldots4}$ pred$_i$

If *SC2* is falsified the cheapest route is advertised by a low-capacity router interface, thus causing an overload.

SC1(**router**: **Router**, **interface**: **Interface**, **rt**: **RoutingTable** **mark**: **Mark**)
pred$_1$: | router.capacity.hasValue("low")
pred$_2$: | interface.router.hasValue(router)
pred$_3$: | rt.gateway.hasValue(interface)
pred$_4$: | rt.marks.contains(mark)
pred$_5$: | mark.ports.!contains(port a)
pre: | $(\wedge_{i=1\ldots4}$ pred$_i) \longrightarrow$ pred$_5 \equiv (\vee_{i=1\ldots4} \neg$ pred$_i) \vee$ pred$_5$

VI. Experimental Setup

Having briefly described the Amazon's network outage scenario and the set of change activities that contributed to it, in this section we discuss how our experimental evaluation is structured, taking advantage of the previously described scenario. We begin our discussion by presenting the model checkers considered for the sake of comparison. Then, we present the IT infrastructure models used for verification, and finally the benchmarks adopted in the evaluation.

A. Model Checkers Used for Evaluation

1) NuSMV Model Checker: NuSMV [10] is a symbolic model checker making use of binary decision diagrams (BDDs). NuSMV supports several different ways to perform model checking: (1) different temporal logics for the specification of safety constraints, Computation Tree Logic (CTL) and Linear Temporal Logic (LTL) [4], [5]; (2) problems can either be model checked using NuSMV's internal binary decision diagram model checker (called binary decision diagram-based model checking herein) or using an external satisfiability (SAT) solver (also referred to as SAT-based model checking). Based on these choices our evaluation experiments benchmark the following optimization techniques offered by NuSMV:

- **Binary decision diagram-based CTL model checking (BDD CTL)**: NuSMV automatically generates binary decision diagrams from the description of the state-transition systems that describe the current configuration and the effects of pending changes. Safety constraints are specified in CTL [4] and verified using NuSMV's internal symbolic model checker.
- **Binary decision diagram-based LTL model checking (BDD LTL)**: Similar to BDD CTL but linear temporal logic (LTL) [4] is used for the specification of the safety constraints.
- **SAT-based LTL model checking (SAT LTL)**: Using SAT LTL model checking, the model checking problem is translated into a satisfiability problem and the external *Minisat SAT solver* is used for verification.

2) SPiN Model Checker: SPiN [8], [9] was originally developed by Gerard J. Holzmann at Bell Labs and has been available as open-source software since 1991. Since then it continues to be adapted to keep pace with developments in the field of formal verification and has been widely used. Different to NuSMV, SPiN is an explicit-state model checker, i.e., it enumerates every state yielded from the execution of effects (change activities) to the initial configuration of the CMDB. The performance of SPiN very much depends on its capability to efficiently store and lookup states during verification which is influenced by the following techniques:

- **No compression (NCP)**: In this technique, states are stored uncompressed. The size of a state/configuration, is the sum of the sizes of all properties of all configuration items in a state.
- **Compression (CP)**: Using compression, every state is compressed before being stored.

TABLE II
Configuration Items (CIs) Used in Each Model. See Fig. 2 for Configuration Items

Model	CIs per server	CIs per network	minimal
$Model_1$	mark1-2, rt1-2	intf1-6, hcr1-2, lcr	not minimal
$Model_2$	mark1-2	not modeled	minimal
$Model_3$	rt1-2	intf1-2,intf5-6	minimal
$Model_4$	mark1-2, rt1-2	intf1-2,intf5-6	minimal
$Model_5$	not modeled	intf1-6	minimal
$Model_6$	not modeled	intf1-6	minimal
$Model_7$	not modeled	intf1-6,hcr1-2,lcr	not minimal
$Model_8$	not modeled	intf1-6,hcr1-2,lcr	not minimal

- **Deterministic minimal automaton (DMA)**: This technique uses minimal automatons to optimize the search. According to Ben-Ari [8], DMA is comparable to binary decision diagram-based model checking in NuSMV. Similar to compression, DMA trades performance for memory consumption.

SPiN always makes use of partial-order reduction, a technique similar to our approach, which excludes certain interleaving of effects by exploiting the commutative characteristics among effects of change activities. The specific requirements of partial-order reduction (among them commutability of effects in respect to the states created [4]) does not hold for all effects of the change-verification logic. Nevertheless, partial-order reduction is activated by default in all experiments involving the SPiN model checker.

3) Our Special Purpose Model Checker: Extended Partial Order Reduction (see Section III) has been prototypically implemented in a special purpose model checker based on the Java programming language. The formulas of safety constraints and effects of change activities are described as executable code that is evaluated over an object-oriented (OO) graph which describes the configuration of a CMDB. This has two advantages: first, the use of objects in an OO language is close to the OO representation frequently present in commercial CMDB systems [23] and in the Common Information Model [24]. Second, frequently used effects and formulas can be natively compiled at runtime (called Just-in-time compilation) by the Java virtual machine to improve the performance of the verification algorithm. Thus, two optimization techniques are considered for the EPOR model checker:

- **Just-in-time compilation (JIT)**: formulas of safety constraints and effects of change activities compiled in machine executable code;
- **No Just-in-time compilation (noJIT)**: formulas of safety constraints and effects of change activities interpreted by Java virtual machine.

B. Infrastructure Models Used for Verification

In summary, the models we used in our benchmarks were basically composed of routers, which in turn could be of high or low capacity. Each of the routers could have a number of routing interfaces, a routing table, and a mark (which is used for routing purposes). An example of a CMDB instance we considered is illustrated in Fig. 2. A brief description of the models we considered is shown in Table II.

TABLE III
RUNTIME COMPLEXITY DEPENDING ON MODEL, WORKLOAD, MODEL CHECKER, AND OPTIMIZATION TECHNIQUE

Benchmrk		OWN configurations		NuSMV configurations			SPIN configurations		
Model	Workload	JIT	noJIT	BDD CTL	BDD LTL	SAT LTL	NCP	CP	DMA
Model_1	$SHTp$	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	$SHTn$	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	FOp	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	FOn	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	$SHTp$ & $SHTn$	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	FOp & FOn	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	$SHTp, SHTn, FOp, FOn$	linear	linear	polyn	polyn	polyn	exp	exp	exp
Model_2	$SHTp$	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	$SHTn$	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	$SHTp$ & $SHTn$	linear	linear	polyn	polyn	polyn	exp	exp	exp
Model_3	FOp	linear	linear	polyn	polyn	polyn	exp	exp	[polyn : exp]
↑	FOn	linear	linear	polyn	polyn	polyn	exp	exp	[polyn : exp]
↑	FOp & FOn	linear	linear	polyn	polyn	polyn	exp	exp	exp
Model_4	$SHTp$	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	$SHTn$	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	FOp	linear	linear	polyn	polyn	polyn	exp	exp	[polyn : exp]
↑	FOn	linear	linear	polyn	polyn	polyn	exp	exp	[polyn : exp]
↑	$SHTp$ & $SHTn$	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	FOp & FOn	linear	linear	polyn	polyn	polyn	exp	exp	exp
↑	$SHTp, SHTn, FOp, FOn$	linear	linear	polyn	polyn	polyn	exp	exp	exp
Model_5	$HCRIncrM$	linear	linear	[polyn : exp]	[polyn : exp]	[polyn : exp]	exp	exp	exp
↑	$LCRDecrM$	linear	linear	[polyn : exp]	[polyn : exp]	[polyn : exp]	exp	exp	exp
↑	$HCRIncrM, LCRDecrM$	linear	linear	[polyn : exp]	[polyn : exp]	[polyn : exp]	exp	exp	exp
Model_6	$HCRIncrOSPF$	linear	linear	[polyn : exp]	[polyn : exp]	[polyn : exp]	exp	exp	exp
↑	$LCRDecrOSPF$	linear	linear	[polyn : exp]	[polyn : exp]	[polyn : exp]	exp	exp	exp
↑	$HCRIncrOSPF, LCRDecrOSPF$	linear	linear	polyn	polyn	polyn	exp	exp	exp
Model_7	$HCRIncrM$	linear	linear	polyn	polyn	polyn	exp	exp	indefinable
↑	$LCRDecrM$	linear	linear	polyn	polyn	polyn	exp	exp	indefinable
↑	$HCRIncrM, LCRDecrM$	linear	linear	polyn	polyn	polyn	[polyn : exp]	[polyn : exp]	indefinable
Model_8	$HCRIncrOSPF$	linear	linear	polyn	polyn	polyn	exp	exp	indefinable
↑	$LCRDecrOSPF$	linear	linear	polyn	polyn	polyn	exp	exp	indefinable
↑	$HCRIncrOSPF, LCRDecrOSPF$	linear	linear	polyn	polyn	polyn	[polyn : exp]	[polyn : exp]	indefinable

The models differ in the configuration items comprised in each CMDB instance (Table II), the workload of change activities supported by each model (Table III), and the specification of change activities and safety constraints in each model. Among the models there are highly optimized ones for a restricted workload (e.g., Model_2 for $SHTp$ and $SHTn$ change activities and Model_3 for FOp and FOn change activities) and models that support workloads comprising a greater variety of change activities (e.g., Model_1 and Model_4 for any static routing change). Furthermore, there are minimal models, i.e., models with the minimal configuration items and properties necessary to verify a workload. For example, Model_4 is the minimal model to verify any combination of static routing change activities, while Model_1 solves the same workloads but contains unnecessary modeling detail (see Table II). Minimal models require more experience in modeling but they also often provide the better performance. Complex models are close to reality but lack verification performance. As it cannot be expected of a change manager to find the most efficient model for verification, we address in this work a broad range of models to solve the same workloads. All models have been carefully engineered in the domain specific language of each model checker to ensure the same detail of models, the same logical specification of change activities and safety constraints, and the best performance for each model checker.

C. Benchmarks

With the eight models and the workloads supported by each of them (see Tables II and III) we obtain 32 combinations of models and supported workloads. Each of these combinations is called a benchmark and can be verified using any of the three model checkers and any of the optimization techniques (see Section VI-A) provided by it. With each three optimization techniques for the NuSMV and SPiN model checkers and two for our proposal, we obtain $8 \times 32 = 256$ runs of benchmarks[4] (shortly called *runs*). In each benchmark the configuration items describing a server (this depends on the model, see Table II) are instantiated and linked to the configuration items describing the network. We look at clusters of size 255, i.e., a group of 255 servers are managed by four high-capacity and two low-capacity router interfaces as depicted in Fig. 2. After that the change activities of the workload are instantiated over every server/network configuration item(s), depending on whether they change the configuration of a server ($SHTp$, $SHTn$, FOp, FOn) or a network (otherwise). For example, consider the benchmark of Model_1 and the workload $\{SHTp, SHTn, FOp, FOn\}$ (see entry in Table III). All change activities in the workload are instantiated over configuration items of the server as they only change the settings of a server. Thus, solving this benchmark on a CMDB comprising 10 000 servers means to verify 40 000 change activities (four change activities per server).

VII. EVALUATION

In this section we present an analysis of the results achieved for each of the model checkers considered: our proposal

[4]All runs were performed on an Intel Xeon Processor with 2.8 Ghz and 4 GB of RAM and the results presented herein comprise all problem sizes that can be solved within 1 GB of RAM and 12 minutes.

TABLE IV
AVERAGE PERFORMANCE FOR EACH MODEL CHECKER DEPENDING ON THE CHOSEN OPTIMIZATION TECHNIQUE

Benchmark		OWN configurations		NuSMV configurations			SPIN configurations		
CMDB size	Position	JIT	noJIT	BDD CTL	BDD LTL	SAT LTL	NCP	CP	DMA
large	fastest	**100%**	0%	37.5%	0%	**62.5%**	25%	**75%**	0%
large	slowest	**0%**	100%	0%	100%	**0%**	0%	**0%**	100%
large	intermediate	**0%**	0%	62.5%	0%	**37.5%**	75%	**25%**	0%
small	fastest	0%	**100%**	**100%**	0%	0%	**81.25%**	28.13%	0%
small	slowest	100%	**0%**	**0%**	100%	0%	**0%**	0%	100%
small	intermediate	0%	**0%**	**0%**	0%	100%	**18.75%**	71.87%	0%
	max size	**81.25%**	68.75%	**56.25%**	18.75%	59.38%	25%	**96.88%**	28.13%
	min size	**18.75%**	31.25%	**9.38%**	81.25%	18.75%	62.5%	**3.13%**	40.63%
	middle size	**0%**	0%	**34.38%**	0%	21.88%	12.5%	**0%**	31.25%

(OWN), NuSMV, and SPiN. We begin by presenting a brief summary of our major findings. Then, we present a more detailed discussion for each of the model checkers we evaluated.

A. Summary

Table IV shows the average performance (over all models and workloads supported by them) for each model checker depending on the chosen optimization technique. For example, SAT LTL is the best choice in terms of performance on large CMDBs in 62.5% of all cases for the NuSMV model checker (measured over all models and workloads).

With regard to performance, our proposal always outperforms NuSMV with linear instead of polynomial and SPiN with linear instead of exponential runtime complexity, regardless of the model, workload, and optimization technique used.

Our proposal was also able to solve larger problem instances than the NuSMV and the SPiN model checker independent of the optimization technique or model used, in time- and memory-constrained verification environments. Consequently, even if an inexperienced change manager chooses the worst model and optimization technique for our proposal, it still solves larger instances than the best choice that can be made for the NuSMV or SPiN model checker.

The results achieved have also shown the robustness of our proposal. For example, a bad choice made in the model and optimization technique of our proposal causes a runtime increase linear in the size of the CMDB, compared to the best case choice. In turn, the maximum penalty increases polynomially (NuSMV) or exponentially (SPiN) when a bad model and optimization technique are chosen for both general purpose model checkers.

More importantly, the results have shown that the worst-case runtime performance of our proposal is better than best-case performance of NuSMV and SPiN. Even if an inexperienced change manager chooses the worst-case combination of infrastructure model and optimization technique for our proposal, this choice is still faster than the best-case choice that can be made for the NuSMV or the SPiN model checker.

Finally, we observed that our proposal shows superior performance over NuSMV and SPiN when it comes to verify the different workloads to have caused Amazon's network outage. Our proposal was proven to be superior in runtime complexity, practical runtime performance, robustness in respect to infrastructure models and optimization techniques, and scalability to large problem instances for various workloads of change activities to have caused Amazon's data center outage.

Having provided a brief summary of the results achieved, in the following sections we describe in more detail the performance of each of the model checkers evaluated, considered the set of benchmarks previously discussed.

B. Performance Results of the Special Purpose Model Checker

Next we summarize the results of our proposal.

- **Our special purpose model checker has linear runtime complexity independent of the optimization technique**: The runtime complexity of our proposal is linear in the size of the Configuration Management Database (CMDB), the number of change activities, and the number of safety constraints. For example, the linear runtime complexity can be observed in Fig. 3 which depicts the log-log graph of the verification time of the $SHTn$ workload on Model_4 depending on the size of the CMDB. In log-log graphs linear runtime complexity can be observed in curves with a slope of one.
- **Our proposal outperforms NuSMV and SPiN in any benchmark**: For every benchmark, our proposal always outperforms NuSMV and SPiN. This even holds when the worst-case optimization technique is chosen for our proposal. For example, this can be observed in Fig. 3, in which the measurements for "Just-in-time (JIT)" and "No Just-in-time (noJIT)" compilation always lie well below all curves of the NuSMV and SPiN model checkers. This holds for the remaining 31 benchmarks as well. We later show in Section VII-G the even stronger statement that the worst-case choice of model and optimization technique made for our proposal still outperforms the best-case choice that can be made for the NuSMV and SPiN model checkers.
- **On large domains just-in-time compilation outperforms code interpretation**: Just-in-time compilation only enhances runtime performance on large CMDB sizes while it imposes a runtime penalty on small CMDBs. This is because the time to compile code does not outweigh the faster execution performance gained by the native execution of effects of change activities and formulas of safety constraints for small problem instances. For example,

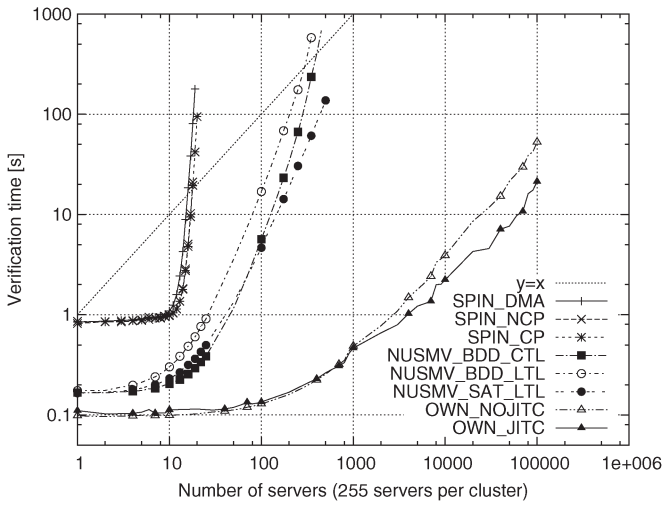

Fig. 3. Verification time of model checkers for $SHTn$ workload on Model_4.

in Fig. 3 noJIT is faster than JIT for problem sizes smaller than 600 servers. After that the runtime improves when just in time compilation is used.

C. Performance Results of NuSMV Model Checker

Next we summarize the results for NuSMV:

- **NuSMV mostly has polynomial runtime complexity**: NuSMV solves 84% (81 out of 96 runs) of the 32 benchmarks in polynomial time, otherwise runtime varies between polynomial and exponential (15 out of 96 runs). The runtime complexity reaches phases of exponential complexity for certain workloads verified with Model_5 and Model_6 (see Table III). The polynomial (or higher/worse) runtime complexity can be observed in Fig. 3 as the Binary Decision Diagram-based Computation Tree Logic (BDD CTL), Binary Decision Diagram-based Linear Temporal Logic (BDD LTL), and Satisfiability Linear Temporal Logic (SAT LTL) curves of the log-log graph have a slope larger than one corresponding to polynomial or exponential complexity. In the corresponding semi-log graphs these curves grow slower than a straight line, which indicates polynomial instead of exponential complexity.

Similar observations can be made for all benchmarks.

- **NuSMV performs better than SPiN and worse than our proposal in any benchmark**: For every benchmark even the worst-case optimization technique of NuSMV outperforms the best-case choice of the SPiN model checker. This can be easily observed in Fig. 3, where the BDD CTL, BDD LTL, and SAT LTL curves always lie well below all measurements made for the SPiN model checker.
- **On large (small) domains SAT LTL (BDD LTL) performs best most frequently**: In 62.5% of all benchmarks SAT LTL model checking provides the best runtime performance on large CMDB instances followed by BDD CTL model checking (best in 37.5% of all benchmarks) and BDD LTL model checking. In turn, on small CMDB instances BDD CTL model checking always performs fastest while the winner for large problem instances (SAT LTL) only provides the second best performance. Similarly to large instances, BDD LTL delivers the worst performance on small instances. Compared to our proposal, the performance of NuSMV can be less easily predicted. While we can obviously recommend JIT for large CMDBs (and, analogously, noJIT for small ones), an obvious recommendation for NuSMV can only be provided for small CMDB instances (BDD CTL model checking) while SAT LTL - the best choice for large CMDB instances - only achieves the best performance in 62.5% of the benchmarks. Note that Fig. 3 provides one of the examples where BDD CTL model checking initially performs better than SAT LTL on small CMDB instances, but is overtaken by SAT LTL model checking when the CMDB size increases.

D. Performance Results of SPiN Model Checker

Next we summarize the results of the SPiN model checker:

- **SPiN mostly has exponential runtime complexity**: SPiN mostly has exponential runtime complexity (85%, 82 out of 96 runs) and for a few cases runtime complexity varies between polynomial and exponential (8%). For some runs (6%) involving Model_7 and Model_8 in combination with the minimal automaton (DMA) optimization technique we were unable to graphically determine runtime complexity because only two problem instances could be solved. The exponential runtime complexity can be identified in Fig. 3 because the slopes of "No compression (NCP)", "Compression (CP)", and "Deterministic minimal automaton (DMA)" curves are larger than one and the curves appear as straight lines in the corresponding semi-log graphs.
- **SPiN always performs worse than NuSMV in every benchmark**: For any benchmark, even the best-case optimization technique for SPiN is outperformed by the worst-case optimization technique of NuSMV. For example, all NuSMV measurements lie well below that of SPiN in Fig. 3.
- **On large (small) domains compression (no compression) performs best on average**: On large domains state compression provides the best runtime performance in 75% of all benchmarks. In the remaining 25% of all cases no compression is faster than compression. In turn, on small domains no compression provides the best performance in 81% of all cases. In the remaining cases compression is faster on small CMDB instances. The deterministic minimal automaton strategy always has the worst-case runtime performance on small and large CMDB instances. On small CMDBs compression causes an overhead that slows down the verification performance such that the "no compression" strategy is faster. However, there are exceptions to this rule. Consequently, the performance of SPiN is less easily to predict than that of our proposal for which optimal choices exist.

E. Maximum Solvable Problem Instances

Besides the runtime and verification complexity, the maximum size of problem instances solvable in a time- and memory-constrained environment is important as well because

significantly sized problem instances can occur for IT change verification. The results are summarized as follows:

- **Our proposal always solves the largest problem instances**: Our special purpose model checker solves the largest problem instances in the time- and memory-constrained environment—even if the best-case optimization technique is chosen for NuSMV and SPiN.
- **NuSMV always solves larger problem instances than the SPiN model checker**: NuSMV always solves larger problem instances than SPiN - even if the best optimization technique is chosen for SPiN.

We now shortly discuss the optimization techniques of each model checker in terms of memory requirements:

- **SPiN: Collapse compression solves the largest problem instances most of the time**: Compression enables the SPiN model checker to solve the largest problem instances in approx. 96.8% of all benchmarks. Compression only has a small impact on the performance because it is also the fastest strategy in 75% of all cases on large CMDBs (see Section VII-D). Consequently, compression is the SPiN strategy that solves the largest problem instances (96.8%) in the shortest time (in 75% of all large CMDBs). Following "Compression (CP)", "Minimal automaton (DMA)" solves the second largest problem instances and "No compression (NCP)" the smallest instances.
- **NuSMV: Binary Decision Diagram-based Computation Tree Logic (BDD CTL) and Satisfiability Linear Temporal Logic (SAT LTL) solve roughly equally large problem instances**: BDD CTL (SAT LTL) solves the largest problem instances in 56% (59%) of all cases. Both can be regarded as roughly equally good when it comes to solve the largest problem instances. However, in 34% of all benchmarks BDD CTL solves the second largest problem instances while SAT LTL only solves 22% of all cases placed second. Consequently, considering first and second places BDD CTL provides the better strategy although it solves slightly less benchmarks placed first. However, BDD CTL is not the best recommendation in terms of performance on large CMDB sizes. Consequently, SAT LTL might still be a good choice. It has slightly worse scalability in terms of problem sizes but its runtime performance is better on large CMDBs (see Section VII-C).
- **Extended Partial Order Reduction (EPOR): Just-in-time compilation (JIT) solves roughly equally large problem instances as code interpretation (noJIT)**: We were unable to determine which optimization technique solves the larger problem instances, as in our experiments the slightly different memory consumptions observed favored neither JIT nor noJIT.

F. Robustness of Runtime Performance

To discuss robustness quantitatively, we introduce the metric described next. The best-case (worst-case) verification time of a workload w using a model checker m is the shortest (longest) runtime for verification that occurs when the optimal (worst) model and optimization technique is chosen to verify w on m. The difference between the worst- and best-case verification time is called penalty. The penalty metric indicates the maximum additional verification runtime that occurs when an inexperienced change manager chooses the most unfavorable model and optimization technique instead of the best solution. Finally, we say that a model checker m_1 is more robust than m_2 with respect to a workload w if and only if the penalty of m_1 (for w) is lower than that of m_2 (for w).

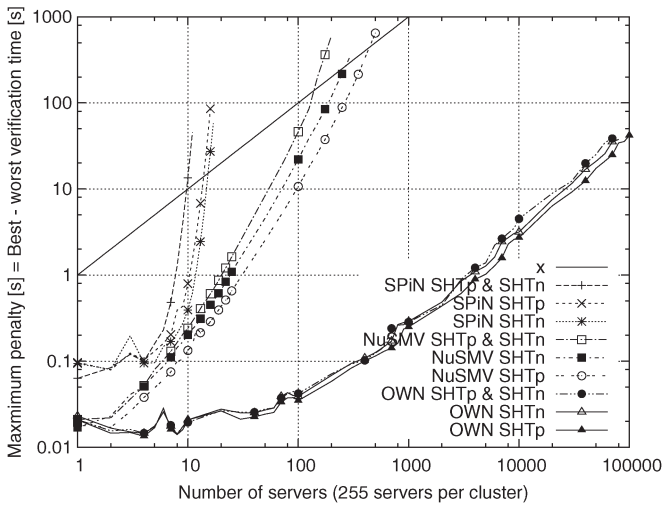

Fig. 4. Maximum penalty of the model checkers on $SHTp$, $SHTn$, and $SHTp \,\&\, SHTn$ workloads.

Fig. 4 exemplary depicts the log-log graph of the maximum penalties of all model checkers for the $SHTp$, $SHTn$, and $SHTp \,\&\, SHTn$ workload. The penalty of our proposal increases linearly with the size of the CMDB as the curves converge to a slope of one in the log-log graph. Thus, should a change manager—due to lack of knowledge in the design of efficient verification domains and models—choose an adverse model and/or optimization technique, he or she will not be penalized more than linearly in the size of the CMDB. In turn, the penalty of NuSMV increases polynomial with the size of the CMDB. Although phases of exponential runtime complexity can be observed for NuSMV and dynamic routing workloads (see Model_5 and Model_6 in Table III), the penalty for these workloads remains polynomial in the size of the CMDB. This is the case because the penalty is only calculated for problem sizes that are solvable in any benchmark, i.e., solvable by every model and optimization technique. As the phases of exponential runtime only appear for problem sizes not solvable in every benchmark, the penalty increases polynomially. Different to NuSMV, the penalty of SPiN is mostly exponential in the size of the CMDB. However, for some workloads ($HCRIncrOSPF$, $LCRDecrOSPF$, and $HCRIncrOSPF \,\&\, LCRDecrOSPF$) the lack of measurements does not allow to determine the complexity of the penalty. All in all, our proposal is not only faster, but also more robust than the SPiN and NuSMV model checkers when it comes to verify the IT change workloads that could have caused Amazon's data center outage.

G. Worst- and Best-Case Performance

Our proposal always outperforms the NuSMV and SPiN model checkers - independent of the model and optimization

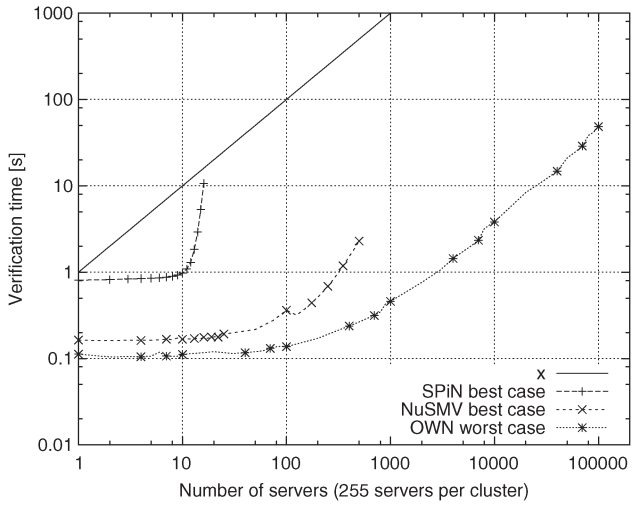

Fig. 5. Worst- vs. best-case verification time of the model checkers on $SHTp$ workload.

technique chosen. Even if an inexperienced change manger chooses the most adverse model and optimization technique for our proposal, it sill outperforms the best choice in terms of model and optimization technique for the other model checkers. In other words, optimization technique and degree of refinement of the infrastructure model do not matter for the EPOR model checker to outperform the others.

For the sake of brevity, we only depict the best- and worst-case performance analysis of the $SHTp$ workload in Fig. 5. The curves of the other workloads have similar shape. The best- and worst-cases are determined for each model checker among all models that support the workload (Model_1, Model_2, and Model_4 for the $SHTp$ workload) and all model checker specific optimization techniques. Note that measurements are only depicted for CMDB sizes for which every combination of model and optimization still solves the verification problem. Thus, we make sure that only problem sizes are considered for which the choice of model and optimization technique does make a difference in terms of computability.

VIII. Conclusion and Future Work

As the success of today's businesses heavily depends on the availability of IT systems and services, every change affecting them comes at a risk that threatens the business. A promising approach to prevent outages caused by IT change operations is the logical verification of changes against safety constraints before their execution. However, existing general purpose model checkers do not scale to the size of realistic configurations. To solve this problem, we introduced in this work Extended Partial Order Reduction (EPOR), a novel model checking algorithm that significantly reduces the complexity of IT change verification from polynomial/exponential to linear complexity. A logic for IT change verification adhering to EPOR has been introduced and was shown to be expressive enough to describe several scenarios that could have caused a recent outage in one of Amazon's data centers. An evaluation comprising a total of 256 runs of 32 benchmarks based on Amazon's data center outage provided evidence that our algorithm significantly outperforms the state of the art SPiN and NuSMV model checkers for a variety of model checker specific optimization techniques, network change operations, and infrastructure configurations. Furthermore, the analysis showed that our approach is more robust to the inputs of change managers inexperienced in the formalization of efficiently verifiable models.

The application of formal methods, e.g, verification, to manage changes and configurations of IT systems, is a relatively new and promising approach to address the challenges imposed on change mangers by the ever increasing complexity of IT infrastructures. It is important to mention that formal methods as presented herein are meant to complement the technical skills of administrators and should find their way into advisory management systems that check administrators' changes for soundness. As a first step of many towards this goal, we introduced in this work a verification approach for IT change verification that, for the first time, possess the scalability necessary to perform IT change verification on large configurations. There are two research challenges we envisage as prospective directions for research in the field. First, processes need to be developed to reliably determine the most critical changes and safety constraints to be protected by verification. We believe that risk analysis is a promising direction to solve this question. Second, methods need to be developed to check the logical correctness and completeness of the formalized change activities and safety constraints.

References

[1] Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region, Apr. 2011. [Online]. Available: http://aws.amazon.com/en/message/65648

[2] S. Lacy and I. Macfarlane, *Service Transition*. Norwich, U.K.: The Stationery Office, 2007.

[3] *ITIL Lifecycle Publication Suite*, The Stationery Office, Norwich, U.K., 2007, Office of Government Commerce.

[4] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[5] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 2001.

[6] S. Kikuchi, S. Tsuchiya, M. Adachi, and T. Katsuyama, "Policy verification and validation framework based on model checking approach," in *Proc. ICAC*, 2007, p. 1.

[7] C. Radu, S. Kikuchi, and M. Kwiatkowska, "Formal methods for the development and verification of autonomic it systems," in *Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development, Verification*, P. Cong-Vinh, Ed. Hershey, PA, USA: IGI Global, 2012, pp. 1–37.

[8] M. Ben-Ari, *Principles of the Spin Model Checker*. London, U.K.: Springer-Verlag, 2008.

[9] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Reading, MA, USA: Addison-Wesley, 2003.

[10] A. Cimatti *et al.*, "Nusmv 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*, vol. 2404. Berlin, Germany: Springer-Verlag, 2002, ser. Lecture Notes in Computer Science, pp. 359–364.

[11] S. Hagen and A. Kemper, "Towards solid it change management: Automated detection of conflicting it change plans," in *Proc. IM*, 2011, pp. 265–272.

[12] S. Hagen, M. Seibold, and A. Kemper, "Efficient verification of it change operations or: How we could have prevented amazon's cloud outage," in *Proc. NOMS*, 2012, pp. 368–376.

[13] R. Calinescu and S. Kikuchi, "Formal methods at runtime," in *Foundations of Computer Software. Modeling, Development, Verification of Adaptive Systems*, vol. 6662, R. Calinescu and E. Jackson, Eds. Berlin, Germany: Springer-Verlag, 2011, ser. Lecture Notes in Computer Science, pp. 122–135.

[14] E. S. Al-Shaer and H. H. Hamed, "Modeling and management of firewall policies," *IEEE Trans. Netw. Serv. Manage.*, vol. 1, no. 1, pp. 2–10, Apr. 2004.
[15] V. C. Hu, D. R. Kuhn, T. Xie, and J. Hwang, "Model checking for verification of mandatory access control models and properties," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 21, no. 1, pp. 103–127, Feb. 2011.
[16] A. Uszok *et al.*, "New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of KAos," in *Proc. IEEE POLICY*, 2008, pp. 145–152.
[17] C. Bartolini, C. Stefanelli, and M. Tortonesi, "SYMIAN: Analysis and performance improvement of the it incident management process," *IEEE Trans. Netw. Serv. Manage.*, vol. 7, no. 3, pp. 132–144, Sep. 2010.
[18] J. P. Sauvé, R. Santos, R. Rebouças, A. Moura, and C. Bartolini, "Change priority determination in it service management based on risk exposure," *IEEE Trans. Netw. Serv. Manage.*, vol. 5, no. 3, pp. 178–187, Sep. 2008.
[19] A. Keller, J. L. Hellerstein, J. L. Wolf, K.-L. Wu, and V. Krishnan, "The champs system: Change management with planning and scheduling," in *Proc. NOMS*, 2004, pp. 395–408.
[20] H. Herry, P. Anderson, and G. Wickler, "Automated planning for configuration changes," in *Proc. LISA*, 2011, pp. 57–68.
[21] S. Hagen *et al.*, "Planning in the large: Efficient generation of it change plans on large infrastructures," in *Proc. CNSM*, 2012, pp. 108–116.
[22] S. Hagen, *Algorithms for the Efficient Verification and Planning of Information Technology Change Operations*, May 2013. [Online]. Available: http://d-nb.info/1037430794/34
[23] A. Keller and S. Subramanian, "Best practices for deploying a cmdb in large-scale environments," in *Proc. IM*, 2009, pp. 732–745.
[24] W. Bumpus, J. W. Sweitzer, P. Thompson, A. R. Westerinen, and R. C. Williams, *Common Information Model: Implementing the Object Model for Enterprise Management*. Hoboken, NJ, USA: Wiley, 2000.
[25] M. Arregoces and M. Portolani, *Data Center Fundamentals: Understand Data Canter Network Design and Infrastructure Architecture, Including Load Balancing, SSL, Security*. New York, NY, USA: Macmillan, 2003.

Sebastian Hagen received the B.Sc. degree in computer science from Augsburg University, Germany, in 2007; the M.Sc. with honours degree in software engineering from Augsburg University, Munich University (LMU), and Technische Universität München (TUM) in 2008; and the Ph.D. degree in computer science from Technische Universität München (TUM). He is a former IBM Extreme Blue Intern (2007) and HP Labs Research Associate (2008–2009). His research interests include the application of formal methods to systems and change management and the automated generation of IT change plans.

Weverton Luis da Costa Cordeiro received the B.Sc. degree from the Federal University of Pará, Brazil, in 2007, the M.Sc. degree from the Federal University of Rio Grande do Sul, Brazil, in 2009, and the Ph.D. degree from the Institute of Informatics of the Federal University of Rio Grande do Sul, Brazil, all in computer science. He is currently an Associate Professor at the Federal Institute of Para at Itaituba-Para, Brazil. His research interests include peer-to-peer systems, network management and security, Information Technology Service Management, and quality of service in wireless *ad hoc*/mesh networks.

Luciano Paschoal Gaspary received the Ph.D. degree in computer science from the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 2002. In 2006, he joined the same institute, where he is now an Associate Professor. In addition to his appointment at UFRGS, he is currently director of the National Laboratory on Computer Networks (LARC) and administrative director of the Brazilian Computer Society (SBC). He is also a member of the IEEE and the ACM. He is author of more than 100 full papers published in leading journals and conferences related to computer networks, network and service management, and computer security. Furthermore, he has been directly involved in the organization and served as technical program committee member of several IEEE, IFIP, and ACM conferences including IM, NOMS, DSOM, IPOM, GLOBECOM, and ICC.

Lisandro Zambenedetti Granville received the M.Sc. and Ph.D. degrees in computer science from Federal University of Rio Grande do Sul (UFRGS), Brazil, in 1998 and 2001, respectively. He is an Associate Professor at the Institute of Informatics of the UFRGS. He has served as a TPC member (2003–2008), General Co-Chair (2004), and Steering Committee member (2005–2008) for the Brazilian Symposium on Computer Networks (SBC/LARC SBRC). Currently, he is member of the Brazilian Internet Committee (CBG.br). He has served as a TPC member for many important events in the area of computer networks, such as IM, NOMS, and CNSM. His main areas of interest include management of network virtualization, Web services and peer-to-peer (P2P)-based management, autonomic computing and self-*, visualization of network management information, and management of Future Internet.

Alfons Kemper received the B.S. degree in computer science at the University of Dortmund, Germany, in 1980, and M.Sc. and Ph.D. degrees from the University of Southern California, Los Angeles, USA, in 1981 and 1984, respectively. From 1984 until 1991 he was an Assistant Professor at the University of Karlsruhe, Germany. In 1991 he became Associate Professor at the RWTH Technical University Aachen, Germany. From 1993 until 2004 he was a Full Professor for Database Systems at the University of Passau, Germany. Starting in 2004 he holds the Chair for Computer Science with a focus on Database Systems at the Technische Universität München (TUM), Germany. From 2005 until 2009 he was the Dean of the CS Department of TUM, thereafter Vice-Dean. His current research focus is in various aspects (query processing, indexing, transaction management, workload management) of main-memory databases.