Towards Solid IT Change Management: Automated Detection of Conflicting IT Change Plans

Sebastian Hagen
Technische Universität München
Department of Computer Science
85748 Garching, Germany
hagen@in.tum.de

Alfons Kemper
Technische Universität München
Department of Computer Science
85748 Garching, Germany
kemper@in.tum.de

Abstract—**Change Management, a core process of the Information Technology Infrastructure Library (ITIL), is concerned with the management of changes to networks and services to satisfy business goals and to minimize costly disruptions on the business. As part of Change Management, IT changes are planned for and scheduled for execution. With several uncoordinated IT operators and different (semi-)automated tools participating in the generation of IT change plans conflicts among them are likely to occur. Furthermore, state of the art IT change planners fail to prevent conflicts among plans. Conflicting IT changes, which render each other infeasible, ultimately lead to failed IT change plans and threaten the continuity of a business due to unsatisfied business goals. To tackle this problem we propose an algorithm for the automated detection of conflicting IT change plans. The algorithm is applied to several IT changes from the network and service management domain. Using simulation we identify and discuss characteristics of IT changes and plans that make deciding conflict freeness among plans more difficult. We find that our algorithm is able to decide the absence of conflicts among synthetically generated IT change plans (1-200 CRs per plan, up to 2000 CRs in total) in the context of large CMDBs (50,000 CIs) and modestly skewed IT changes. The advantage of our solution lies within the tight integration of object-oriented models, frequently used to describe CMDBs in research and commercial systems. Furthermore, existing IT change planners and schedulers remain unchanged while our solution prevents inter-plan conflicts.**

I. Introduction

Change Management [1], a core process of the *Information Technology Infrastructure Library* (ITIL) [2] ensures that changes to hardware and software are managed and conducted in a way that costs are met, risks are reduced, and that the business needs and goals of a company are satisfied with the highest degree of confidence and optimization. To ensure this, ITIL proposes a Change Management process comprising the evaluation, authorization, planning, test, scheduling, implementation, documentation, and review of IT changes [1].

IT Change Planning, an important step of the Change Management process, has been subject to intensive research [3], [4], [5], [6], [7], [8]. This let to IT change planners which can generate a single sound plan that, when executed, achieves a Request for Change. However, all approaches fail to prevent conflicts among IT changes from different plans, that can render IT change plans infeasible. There are several reasons why these conflicts occur: (1) The planners that have been proposed so far are unaware of changes conducted manually outside their visibility, e.g., by an operator. Thus, manually planned changes can render automatically planned changes infeasible and vice versa. (2) Previous research on IT change planning neglects the effects that scheduled, but not yet executed IT changes have on the feasibility of IT change plans currently being planned for. Often several weeks pass in practice until scheduled IT changes are executed [9]. Until then, their effects are unseen by the current planners facilitating the generation of infeasible IT change plans due to false assumptions. (3) In a practical Change Management environment changes are less automated and uncoordinated operators and tools, e.g., from different management domains, can generate plans rendering each other infeasible.

Aware of the threats infeasible IT change plans have on the continuity of an IT business and the satisfaction of business goals, ITIL [1] proposes the implementation of a single point for changes to minimize the likelihood of conflicting changes. However, it remains open how conflicting IT change plans can be automatically detected except by elaborate human inspection.

To aid in the detection of conflicting IT changes, we introduce the theoretical foundations and an algorithm to detect them over object-oriented (OO) *Configuration Management Databases* (CMDB). A conflict detection layer using the proposed algorithm is added in between legacy planners and schedulers only admitting conflict-free IT change plans from uncoordinated sources, such as IT operators and automated planners, to the scheduling engine. Thus, as long as the scheduler respects the precedence constraints of each plan, plans are guaranteed to not render each other infeasible. To show the applicability of our approach to Change Management, the algorithm is applied to conflicting IT changes from the network and services management domain. Using simulation we identify and discuss unfavorable characteristics of IT changes and plans that make them costly to be checked for conflict freeness. Among others, we find that a workload of plans comprising CRs that are skewed over the CIs of a CMDB or that address many CIs are costly to verify. Furthermore, we find that the proposed solution is able to decide the absence of conflicts among synthetically generated IT change plans (1-200 CRs per plan, up to 2000 CRs in total) in the context of large CMDBs (50,000 CIs) and modestly skewed IT changes within reasonable time.

The advantage of our approach lies within the tight integration of *object-oriented* (OO) *Configuration Management Databases* (CMDBs), frequently used by commercial CMDB systems [10] and suggested by the *Common Information Model* (CIM) [11] to model the current state of software and hardware in a data center. In addition to that, previously proposed work on IT change planning [3], [4], [5], [6], [7], [8] and scheduling [12], [13] remains unchanged while inter-plan conflicts are prevented.

The remainder of this work is organized as follows: In Section II we discuss related work and its shortcomings with respect to the occurrence of conflicting IT changes. Section III introduces the architecture and theory to detect conflicting IT changes, followed by our Algorithm in Section IV. We evaluate our solution in Section V using simulation experiments. Finally, Section VI concludes the work.

II. Related Work

Different aspects of IT Change Management, such as planning, scheduling, rollback, and risk assessment, have been addressed in the last recent years. However, despite the negative influence of conflicting IT changes regarding the feasibility of IT change plans - to the best of our knowledge - nobody has yet proposed an approach to detect them. Early work on IT change planning [3], [14], [15] comprises approaches that do not apply logically sound reasoning to IT changes. The plans generated by these approaches are not guaranteed to be executable from a logical point of view. For example, Keller et al. [3] proposed CHAMPS which formalizes planning and scheduling as an optimization problem which achieves a high degree of parallelism. While CHAMPS can reason about dependencies to achieve actions, it does not apply logically reasoning and cannot detect conflicting IT changes among different plans. Cordeiro et al. [14], [15] propose an approach focusing on the reuse of knowledge in IT change design. The authors propose an algorithm to refine abstract IT changes without giving logical guarantees about the feasibility of the generated plans. More recent works, such as [4], [5], [7], including our own research [6], [8], propose approaches for IT change planning that reason about the precondition and effects of IT changes within a single plan. For example, the works by Cordeiro et al. [5] and Trastour et al. [7] focus on the refinement of IT changes, while our hybrid approach [6] addresses refinement and state-based reasoning at the same time. Common to all recent works on IT change planning is, that they are capable of generating a single, logically sound plan as long as the datacenter does not change in between plan generation and execution. However, after a plan is generated, neither of the approaches takes the effects that the plan's scheduled, but not yet executed, IT changes have on the feasibility of later planned IT changes into account. Thus, current approaches cannot prevent that automatically planned IT changes render scheduled and not yet executed changes infeasible and vice versa. In addition to that, they cannot cope with conflicts induced by multi-operator change environments. To cope with already occurred conflicts, we proposed in an earlier work [9] an approach to render infeasible IT change plans feasible again. Compared to that work, the algorithm proposed herein proactively detects conflicts that render IT change plans infeasible.

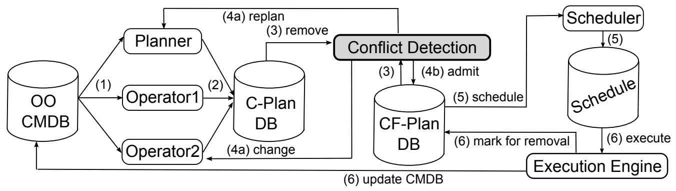

Fig. 1. Intermediate Conflict Detection layer in between IT change planners and schedulers to avoid inter-plan conflicts in a multi-operator environment.

Regarding the scheduling of IT changes Rebougas et al. [12], among others, propose an approach to schedule IT changes into change windows to minimize the costs imposed by violated Service Level Agreements. Zia et al. [13] discuss mixed-integer programming for change scheduling. Common to all these works and others on scheduling is that changes are scheduled into change windows taking precedence constraints within a plan into account. However, previously introduced works on planning do not take into account the effect scheduling decisions have on the feasibility of subsequent plans.

Besides planning and scheduling, Machado et al. [16], [17] propose a rollback solution to deal with failures during change implementation in a reactive way by undoing partially executed change plans. Compared to this work, our approach proactively detects conflicting IT changes and prevents their admission to the change scheduler and execution engine. Wickboldt et al. [18], [19] propose a solution for the automated risk assessment of IT change plans to proactively treat risks during deployment. Similar to this work, our approach proactively avoids failed change plans as well. However, we avoid failed change plans by logically sound reasoning whether changes conflict with each other whereas Wickbold et al. apply risk analysis. Recently, Lunardi et al. [20] discussed the alignment of IT change plans to business objectives and strategies to assign human resources to IT changes [21]. Our work complements these works because it guarantees the feasibility of IT change plans in multi-operator environments and for automated planners - a prerequisite to successfully align IT changes with business objectives or to optimize their assignment to operators.

III. Detection of Conflicting IT Changes

A. General Architecture

Figure 1 depicts the conceptual change planning and scheduling architecture our solution integrates in. Our architecture is build-upon object-oriented modeling techniques. A CMDB is an object-relational model describing the current state of the data center, i.e., all its physical and virtual resources together with its hosted software. The CMDB is implemented as in main memory objects of Groovy [22] a dynamic object-oriented language based on Java. We chose the object-oriented representation because according to Keller et al. [10] today's CMDBs follow an object-oriented approach, sometimes derived from the *Common Information Model (CIM)* [11]. In addition to that, object-oriented models have been used in previous research by Eilam et al. [23] to describe the state of a data center. IT Change plans are generated by different uncoordinated IT operators or by automated IT change planners [3], [6], [8], [4], [5], [7] based on the current state of the OO CMDB (Step 1). Generated plans are stored in a *conflicting plan database* (C-plan DB), i.e., a database of unscheduled, but not yet verified to be conflict-free, plans in Step 2. The *Conflict Detection* layer removes a plan pl from the C-plan DB (Step 3) and checks whether the database of conflict-free plans (CF-plan DB) remains conflict-free if pl is added to it. In case it does not remain conflict-free, the plan is returned to the operator or the planner for adaptation/re-planning, e.g., using our solution previously introduced in [9] (Step 4a). In case the CF-plan DB remains conflict-free, pl is added to it in Step 4b. The scheduler schedules not yet scheduled plans in the CF-plan DB (Step 5). Scheduled, but not yet executed plans, remain in the CF-plan DB. Once the plan is executed by the *Execution Engine* in Step 6, the effects of the IT changes are propagated to the OO CMDB and the executed plan pl is marked for deletion in the CF-Plan DB. However, pl remains in the CF-plan DB as long as the C-plan DB contains plans that were planned before pl was executed, because these plans were not planned taking the effects of IT changes in pl into account. Plan pl still has to be taken into account when admitting these plans to the CF-plan DB.

B. Definitions:

The object-oriented CMDB is described by a set $M = \{o_1, ..., o_n\}$ of Java objects. Each object o_i has properties $prop(o_i) = \{o_i.p_1, ..., o_i.p_n\}$ denoted by their full qualified path consisting of an object and the name of the property. An IT change plan pl is a 2-tuple $pl = (CR_{pl}, <_{pl})$ such that $CR_{pl} = \{cr_1, ..., cr_n\}$ are the CRs of pl and $<_{pl} \subseteq CR_{pl} \times CR_{pl}$ a partial precedence order among CR_{pl} describing *Finish to Start* constraints regarding valid execution sequences of pl. In the context of this work a *change request* (CR) cr is the description of an IT change, in particular its' precondition and effects. More formally, a change request cr is a tuple $cr = (d_{cr}, pre_{cr}, eff_{cr})$ where d_{cr} is the textual description of cr and pre_{cr} a precondition. A precondition is implemented as a boolean expression evaluated over reads of object properties from the CMDB. pre_{cr} needs to account in order to execute cr, i.e., to apply its effects eff_{cr}. Effects are implemented as dynamically executable code which reads and writes the properties of objects, i.e., CIs, in the CMDB. For $x \in \{pre_{cr}, eff_{cr}\}$, we denote by $R(x) \subseteq \bigcup_{o \in M} prop(o)$ or $W(x) \subseteq \bigcup_{o \in M} prop(o)$ the set of all properties of objects read or written by the precondition or the effects of cr. Note, that $W(pre_{cr}) = \emptyset$ accounts because only eff_{cr} changes CIs in the CMDB. Based on the read and writes of IT changes to properties of CIs / objects in the CMDB, a conflict is defined

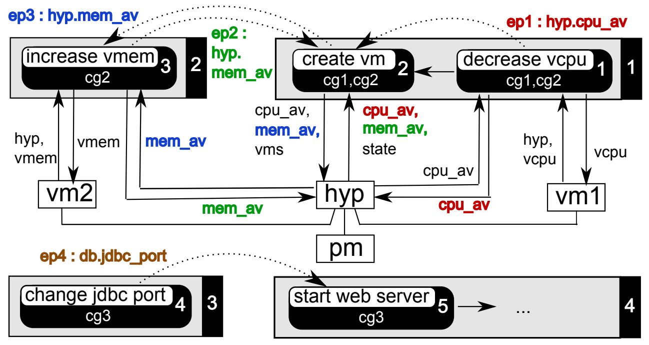

Fig. 2. Conflicting IT changes around the creation of a virtual machine

as follows:

Definition 3.1: Effects-Precondition (EP) - conflict
Two CRs cr_1, cr_2, $cr_1 \neq cr_2$ are in an ordered EP-conflict ($cr_1 <_{EP} cr_2$) regarding object $o \in M$, iff $\exists p \in prop(o)$: $o.p \in W(eff_{cr_1}) \cap R(pre_{cr_2})$. We denote by $<_{EP} \subseteq CRs \times CRs$ a relation describing the EP-conflicts among all CRs of all plans. Thus, $(cr_1, cr_2) \in <_{EP}$ iff cr_1 writes a property of an object that is also read by cr_2. This means that the execution of cr_1 can, but must not necessarily, render cr_2 infeasible. Note, that $<_{EP}$ is not reflexive and can be symmetric, transitive, or cyclic.
To capture which CRs cannot be verified independently of each other due to chains of EP-Conflicts or several conflicts on the same CI, the notion of a Conflict Group is introduced:

Definition 3.2: Conflict Group
For a set of plans $\{pl_1, ..., pl_n\}$ let $G = (CRs, <_{EP})$ be a directed graph such that $CRs = \bigcup_{i \in \{1,...,n\}} CR_{pl_i}$ are the nodes and $<_{EP} = \bigcup_{i \in \{1,...,n\}} <_{pl_i}$ the edges of G. A conflict group is a connected component in graph G' emerging from G when the direction of edges is neglected. More formally, $G' = (CRs, <'_{EP})$ where $(cr_i, cr_j) \in <'_{EP}$ iff $(cr_i, cr_j) \in <_{EP}$ or $(cr_j, cr_i) \in <_{EP}$. Thus a CG cg is a set of CRs $\{cr_1, ..., cr_k\}$, $k \in \mathbb{N}$ with two important criteria: (C1) Each $cr \in cg$ is only part of one conflict group. (C2) $\forall cr_i \in cg : \neg\exists cr \notin cg$: $(cr, cr_i) \in <_{EP} \vee (cr_i, cr) \in <_{EP}$, i.e, the conflict group is closed with respect to Effect-Precondition conflicts in both directions.
A CG defines the smallest set of CRs which participate in conflicts that cannot be independently verified of each other. The following subsection provides an example.

C. Example

Consider plans pl_1 and pl_2 (grey boxes) in Fig. 2. Plan pl_1 consists of cr_1 (*decrease vcpu*) and cr_2 (*create vm*). The precedence constraints $<_{pl_i}$ of plan pl_i are denoted by non-dashed arrows in between CRs. In the given example $(cr_1, cr_2) \in <_{pl_1}$. The underlying OO CMDB consists of 4 CIs / objects: A hypervisor (hyp) which runs on a physical machine (pm) and manages two virtual machines ($\{vm_1, vm_2\}$). Arrows among CRs and objects describe the read and writes to properties of objects conducted by preconditions and effects. For example, cr_1 decreases the number of virtual cpus allocated to virtual machine vm_1. Thus, cr_1 reads and writes property $hyp.cpu\_av$ which describes the number of not yet allocated cpu resources by the hypervisor. cr_2 (*create vm*) reads this property in its precondition from the hypervisor object in the CMDB to check whether enough cpu resources are available to create a new VM. Thus, the increase of $hyp.cpu\_av$ has the potential to influence the creation of a new VM on the same host. This is captured by EP-conflict $ep_1 = (cr_1, cr_2)$ (red) in Figure 2. In fact, because cr_1 is ordered before cr_2 we expect cr_1 to be even necessary to successfully create the VM. Otherwise, $(cr_1, cr_2) \in <_{pl_1}$ would be obsolete. Note, that $(cr2, cr1) \in <_{EP}$ accounts as well (also due to an EP-conflict regarding $hyp.cpu\_av$). However, because cr_2 is ordered after cr_1, cr_2 does not have the chance to render cr_1 infeasible, because every valid execution sequence of pl_1 will have cr_1 ordered before cr_2. More precisely, EP-conflicts ordered in the opposite direction of the transitive closure of precedence constraints of a plan ($<_{pl_i}$) do not need to be taken into account to decide the feasibility of a CR. There are also EP-conflicts between cr_3 (*increase vmem*) in pl_2 and cr_2 (*create vm*) in pl_1 in both directions because the increase of the RAM of vm_2 can make the creation of a VM on the same host infeasible due to insufficient RAM and vice versa. Thus, $ep_2 = (cr_3, cr_2) \in <_{EP}$ (green) and $ep_3 = (cr_2, cr_3) \in <_{EP}$ (blue) due to reads and writes to $hyp.mem\_av$ (see Fig. 2).

IV. The Algorithm

Algorithm 1 Check new plan for absence of conflicts

```
 1: procedure CHECK_PLAN(Plan pl)
 2:     verify_top_order(get_top_order(pl), true)         /* see Algorithm 2 */
 3:     grps_to_verify = []
 4:     for all conf ∈ ConflictManager.EP_Conflicts do
 5:         grp = create_or_merge(conf.from,conf.to)
 6:         grps_to_verify.add(grp)
 7:     end for
 8:     for all grp ∈ grps_to_verify do
 9:         if !verify_cg(grp) then return false end if   /* see Algorithm 3 */
10:     end for
11:     return true
12: end procedure
```

Algorithm 2 Verification of a topological order

```
 1: procedure VERIFY_TOP_ORDER(TopOrder order, boolean log_conflicts)
 2:     objects_changed = []
 3:     for all cr ∈ order do
 4:         if cr.executable() then
 5:             cr.apply_effects()
 6:             objects_changed.addAll(cr.objects_changed)
 7:             if log_conflicts then ConflictManager.add_conflicts(cr) end if
 8:         else
 9:             restore_objects(objects_changed)
10:             return false
11:         end if
12:     end for
13:     restore_objects(objects_changed)
14:     return true
15: end procedure
```

For n plans in the CF-plan DB (see Fig. 1), previously verified to be conflict-free, Alg. 1 returns true if the CF-Plan DB remains conflict-free if pl is added. Consider plans 1-4 in Fig. 2. Note, that reads and writes of pl_3 and pl_4 are not shown to keep matters simple. Please refer to Fig. 3 for reads and writes of these CRs to the CMDB. Assume pl_1 is the first plan to be checked for absence of conflicts by Alg. 1. Line 2 in Alg. 1 determines a topological order in linear time $O(|CR_{pl_1}| + | <_{pl_1} |)$ and passes it to Alg. 2. In case of pl_1, Alg. 2 is called with topological order $< cr_1, cr_2 >$, the only topological order of pl_1. Algorithm 2 has two purposes: (1) Check whether the execution sequence of a topological order of CRs is valid in the sense that the execution of a CR does not render a subsequent one in the topological order infeasible. (2) Determine the EP Conflicts among the CRs in the topological order if called with *true* in the second argument. Thus, Alg. 2 executes the chosen topological order of pl_1 (Lines 3–12). The precondition of each CR is checked (Line 4) and the following happens if it accounts: (1) The effects of the change request are applied to the model (Line 5) (2) all objects changed by the effects of the CR are accumulated in a helper variable (Line 6) and (3) the conflicts cr participates in are reported to a Conflict Manager, later queried in Alg. 1, Line 4. Conflicts are logged by intercepting reads and writes to properties and storing them in an index structure. Thus, it can be decided in respect to previous reads and writes whether an EP-conflict occurs among CRs. If a CR in the topological order is not executable or if all CRs were executed successfully all objects altered in the CMDB need to be restored in Line 9 or 13 . For example, the original values of properties $vm_1.vcpu$ and $hyp.cpu\_av$ need to be restored because they were changed by the effects of CR *decrease vcpu* (see writes in Fig. 2). This is achieved by restoring previously backed up values of the properties of the affected objects. We found this solution to be the fastest among others in [8]. Line 7 in Alg. 2 also reports conflict ep_1 logged among CRs *decrease vcpu* (cr_1) and *create_vm* (cr_2) respectively property $hyp.cpu\_av$ to a conflict manager. If all CRs could be successfully executed true is returned in Line 14. Alg. 1 continues with processing all logged conflicts that CRs of plan pl (pl_1) participate in Lines 4–7. The CRs ($\{cr_1, cr_2\}$) of the previously logged conflict (ep_1) are passed to the *create_or_merge()* method behaving as follows: If the CRs inducing a conflict are not yet part of a CG, a new dedicated CG is returned only containing the CRs of the conflict. Otherwise, at most two distinctive CGs need to be merged together in $O(max(|cg_i|, |cg_j|))$ to keep the CGs closed as they are the connected components in the undirected conflict graph (see Definition 3.2). In case of ep_1 a new CG (cg_1, see Fig. 2) is created only comprising

Algorithm 3 Verification of a conflict group

```
1: procedure VERIFY_CG(ConflictGroup cg)
2:     top_orders = all_topological_orders(cg)
3:     for all top_order ∈ top_orders do
4:         if !verify_top_order(top_order, false) then return false end if
5:     end for
6:     return true
7: end procedure
```

cr_1 and cr_2 because none of them is yet part of another CG. Algorithm 1 iterates through all newly created or modified CGs (in our case only cg_1) in Lines 8–10 and returns true if all groups could be verified (Line 11) by Alg. 3. Otherwise, false is returned in Line 9. Algorithm 3 verifies in factorial time $O(|cg| * |cg|!)$ whether a conflict group cg is conflict-free by calculating all topological orders in respect to the temporal constraints of the CRs in a conflict group in Line 2. For each topological order Alg. 2 is called to verify whether this is a proper execution order in which no CR renders a subsequent one infeasible. However, this time reads, writes, and conflicts are not logged. For a conflict group cg and a change request cr, $cr \in cg$, all other CRs from any plan that might render cr infeasible or all other CRs that cr might render infeasible are contained in the same CG by construction (see Def. 3.2-(C2)). By checking each topological order of CRs in a CG we can guarantee that none of the EP-conflicts renders a CR infeasible in practice. All in all, for cg_1 only topological order $< cr_1, cr_2 >$ needs to be checked. Assume plan pl_2 (Fig. 2), only consisting of CR *increase vmem* (cr_3), arrives next. The execution of the topological order of pl_2 (Alg. 1, Line 2) yields EP-conflicts ep_2 and ep_3 over property $hyp.mem\_av$ (see reads and writes in Fig. 2). The previously created group $cg_1 = \{cr_1, cr_2\}$ is not *closed* under $<_{EP}$ (see Definition 3.2-(C2)) any more due to ep_2 and ep_3. Thus, Line 5 in Alg. 1 merges cr_3 into cg_1 forming cg_2 comprising CRs 1, 2, and 3 (see Fig. 2) which is added to the set of CGs to verify in Line 6. Note, that nothing needs to be done for ep_3 in Line 5 because the CRs of ep_3 are already part of the same CG (cg_2). To verify cg_2, Alg. 2 calls Alg. 3 for each topological order $< cr_1, cr_2, cr_3 >$, $< cr_1, cr_3, cr_2 >$, and $< cr_3, cr_1, cr_2 >$ of cg_2 in Line 4. When pl_3 is to be checked no EP-conflicts are logged. However, when applying the effects of cr_4 a write to property $jdbc\_port$ of object db which sets the new configuration of the port used by the database for incoming JDBC connections is recorded in Alg. 2, Line 5. When pl_4 arrives, a conflict between *start web server* (cr_5) and cr_4 is detected regarding property $db.jdbc\_port$ because it is also read by cr_5 in its precondition to check whether the JDBC port it tries to connect to matches the port the database is listening on (compare equivalent CR in Fig. 3). Thus, a new EP conflict (ep_4, brown) is detected in Alg. 2, Line 7. As a consequence a dedicated CG for CRs 4 and 5 is created (cg_3). To verify whether cr_4 renders cr_5 infeasible, it suffices to verify cg_3. Note, that cg_2 does not need to be verified again because it was previously verified and did not change by the arrival of pl_3 and pl_4. Conflict groups partition the set of all CRs of all plans in disjunct connected components of CRs independently verifiable. This makes the automated detection of conflicting IT change plans computationally feasible because not all possible interleaved execution sequences of all plans need to be checked. Algorithm 1 is logically sound, i.e., no false-positives or false-negatives occur. It detects without overlook whether a plan conflicts with a previous plan and only reports a plan as conflicting if it can render another CR infeasible for a particular interleaved execution sequence.

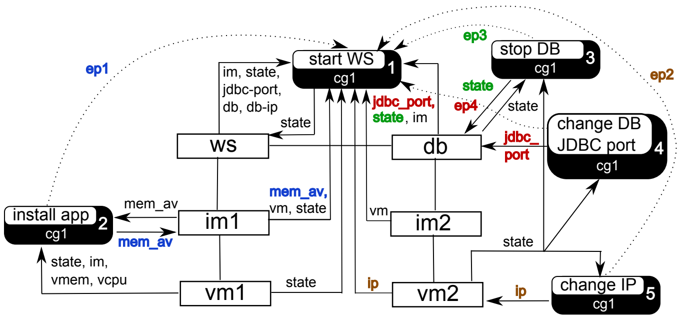

Fig. 3. Conflicting IT changes around a CR to start a web server

V. Evaluation

A. Conflicting IT Change Examples

To show the applicability of the proposed solution to detect conflicting IT changes, different IT changes were described by preconditions and effects and were used with the algorithm. Among these the problems to decide conflict freeness among the IT changes and plans around a CR to create a VM in Fig. 2. To verify whether plans pl_1 and pl_2 are conflict-free, conflict group cg_2 (see Fig. 2) needs to be verified. cg_2 has three topological orders and its verification takes 23ms. Figure 3 depicts another set of conflicting IT changes evolving around cr_1, a CR to start a web server. We briefly discuss the conflicts modeled using our technique: **(Conflict 1)** cr_2 installs an application on image im_1, reducing the available memory on the image to $im_1.mem\_av$. When cr_1 starts the web server, pre_{cr_1} reads the property to check for enough swap space to start the web server. Thus, cr_2 can render cr_1 infeasible (see ep_1, blue). **(Conflict 2)** cr_5 changes the IP address of vm_2, the VM the database runs on, by writing to property $vm_2.ip$. However, this property is read by pre_{cr_1} to check whether $vm_2.ip$ matches the IP address the web servers delegates its JDBC queries to ($ws.db - ip$). If they do not match, the web server cannot be started (see ep_2, brown). **(Conflict 3)** cr_3 stops the database the web server relies on. Thus, cr_1 cannot start the web server. This is modeled by an EP-conflict regarding the $state$ property of db (see ep_3, green). **(Conflict 4)** cr_4 reconfigures the JDBC port the database listens to for incoming JDBC connections by writing property $db.jdbc\_port$. The property is read by pre_{cr_1} to check whether the JDBC port used by the web server ($ws.jdbc\_port$) matches the port of the database's JDBC daemon. If the ports do not match, cr_1 cannot be executed (see ep_4, red, Fig. 3).
Whether two plans conflict can usually be decided faster than the absence of conflicts because Alg. 1, Line 9 returns false as soon as a non-executable topological order of a CG is found. For example, verifying CRs 1, 2, 4, and 5 (Fig. 3) for absence of conflicts (cr_4 and cr_5 change the port / ip address to the same value) takes 40ms, while a conflict can be found between 4 to 25ms, depending on whether which topological order fails. Conflict detection in cg_1 containing all CRs in Fig. 3 takes between 4.3ms and 43ms.

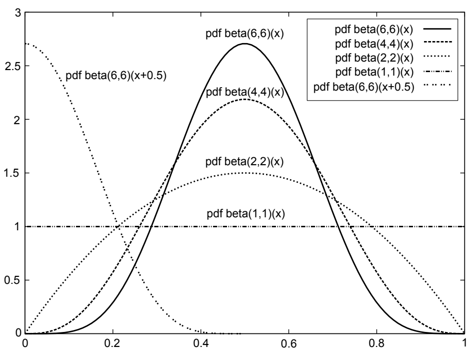

Fig. 4. PDF of Beta distributions for $\alpha = \beta \geq 1$

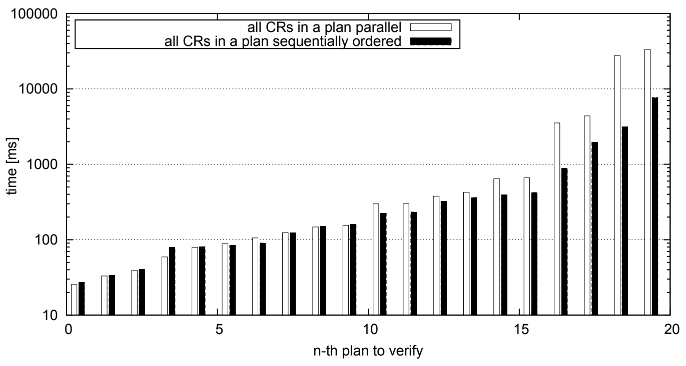

Fig. 5. Time to verify n-th plan depending on temporal ordering of CRs within n-th plan. 50,000 CIs, 2000 CRs, plan length $U(1, 200)$, medium skewed CRs ($sp_{ci} = 4$), $sp_{rw} = 6$, $max_{rw} = 4$, and $max_r = 3$

B. Simulation Experiments

To analyze the characteristics of IT changes that make IT change plans costly to verify, simulation experiments are conducted. All measurements presented in this work are done on a Core2 Duo 2.8Ghz machine with 6MB Cache and 4GB of RAM. We take into account, that some CIs in the CMDB might be subject to more IT changes, i.e., writes and reads of CRs, than others. For example, some CIs might have a higher maintenance effort or are subject to IT changes by different IT practitioners, e.g., network and hypervisors. We use Beta distributions to describe several kinds of skew that might be inherent to IT change plans. Beta distributions are a family of continuous probability distributions defined on the open interval (0,1) and parameterized by two positive shape parameters α and β. In the special case of $\alpha = \beta = 1$ the Beta distribution equals the continuous uniform distribution over (0,1) (see Fig. 4). For $\alpha = \beta > 1$ the PDF becomes bell shaped (comparable to the normal distribution), modeling hot-spots around its expected value $E(x) = \frac{\alpha}{\alpha+\beta} = \frac{1}{2}$ (see Fig. 4). Note, that Beta distributions have a finite domain (0,1) while normal distributions have an infinite domain preventing their discretization to CIs in the CMDB. In the remainder of this work we use Beta distributions with $\alpha = \beta = sp$ and refer to sp as their distinct shape parameter. A Beta distribution (see Fig. 4), discretized over all CIs in the CMDB, i.e., over interval $[0, |CMDB| - 1]$, with shape parameter sp_{ci} is used to model the likelihood that a CI is addressed by a read / write of a CR. For the example CRs it can be observed that most of them do not read and write (RaW) more than 1-2 properties at the same time. Thus, another Beta distribution with shape parameter sp_{rw}, shifted to the left by 0.5 on the x-axis, and discretized to values in $[1...max_{rw}]$ is used to account for the number of attributes read and written at the same time by a CR (see Fig. 4). A few CRs, such as *start WS* (see cr_1 in Fig. 3), rely on additional reads of properties (issued by the precondition to several additional CIs) that are not written by the same CR. We use the same Beta distribution as for simultaneous reads and writes (sp_{rw}) to model the number of additional reads. However, discretized to a different interval ($[1, max_r]$). A uniform distribution $U(1, pl_{length})$, where pl_{length} is the maximum length of a plan, describes the lengths of plans. Note, that the purpose of the simulation is to generate a large number of CRs and plans engaging in thousands of conflicts to determine the characteristics of plans and CRs costly to verify. To achieve realistic times to evaluate a precondition and to apply the effects of a CR, a uniformly chosen CR from the example CRs is checked / applied as well with every CR of the simulation to a second CMDB. However, the additional preconditions and effects are not taken into account regarding EP-conflicts. Thus, CRs are simulated with realistic complexity in respect to the precondition and effects evaluation but with the proper characteristics for the experiment.

C. Influence of plan features on verification time

1) ***Temporal constraints among CRs in a plan:*** The more temporally ordered CRs of a plan are, the more likely an intra-plan conflict appears among ordered CRs. For example, consider cr_1 and cr_2 of plan pl being in conflict to each other and thus in the same conflict group cg. If $cr_1 <_{pl} cr_2$ or $cr_2 <_{pl} cr_1$, the number of topological orders of cg is reduced by the factor of 2 compared to the parallel case. Thus, highly temporally ordered plans have the potential to reduce the number of topological orders of a conflict group. To notice this effect, plans need to be sufficiently long and conflicts need to be likely. For example, Fig. 5 depicts the times to verify a sequence of 20 plans whose length is uniformly distributed between 1 and 200 CRs for two different types of ordering constraints: (1) All CRs within a plan are parallel and (2) the CRs of each plan are sequentially ordered. It can be observed that the difference between ordered and unordered plans has a stronger impact on the plans towards the end of the sequence when conflict groups become larger and conflicts more likely. We conclude that the more ordered the CRs of a plan are, the less effort is necessary to verify it compared to the same plan comprising less ordering constraints.

2) ***Length of plan:*** Assuming that n plans have already been successfully checked for conflict freeness, the $(n + 1)$-st plan will take the longer the more CRs it comprises (assuming equal likelihood of conflicts). A longer plan can engage in more conflicts with previous CRs leading to more and eventually larger CGs to verify. However, this does not necessarily

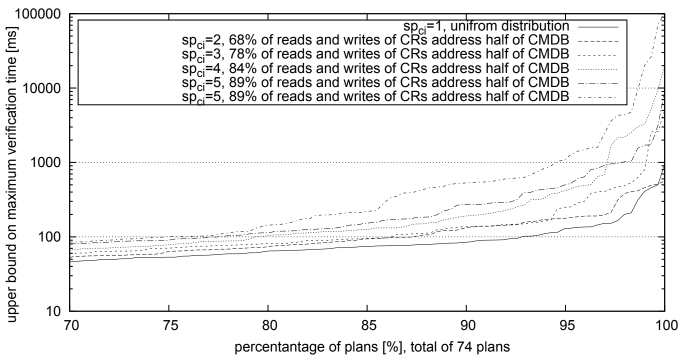

Fig. 6. Quantile of the time to verify n-th plan depending on the skew of IT changes over the CMDB. CMDB comprises 50,000 CIs, approx. 2000 CRs, plan length $U(1, 50)$, 50% of all CRs in a plan transitively orderd, $sp_{rw} = 6$, $max_{rw} = 4$, and $max_r = 3$

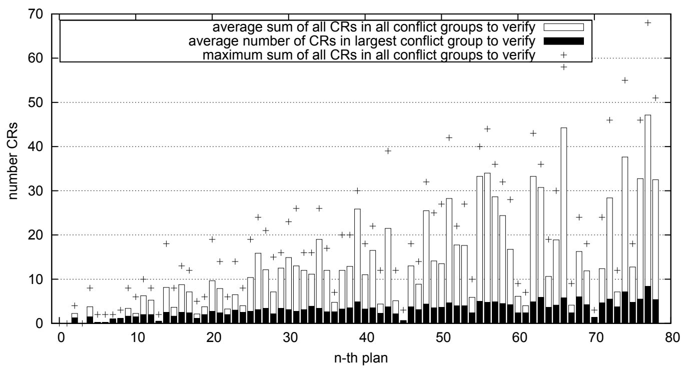

Fig. 7. Average sizes of CGs when verifying several randomly generated sequences of plans. Setup as in Fig. 6 but strongly skewed CRs ($sp_{ci} = 6$)

account for long plans occurring early in a sequence of plans to verify because the likelihood of conflicts can be small at the beginning. The later a long plan appears in a sequence of plans to verify, the bigger its disadvantage in length will be. The shorter plans are, the less conflicts they engage in (assuming equal likelihood of conflicts between CRs) which can lead to decent verification times even if many plans were processed before (see Paragraph V-C5)

*3) **Read / write skew of CRs over CIs**:* The more skewed reads and writes of CRs are over the CIs of the CMDB, the more likely EP-conflicts become. This results in larger conflict groups, whose size factorially influences the runtime of the proposed solution. Figure 6 depicts the worst case runtime for the upper 70-100% of plans depending on the skew the CRs have over the CMDB. Skew is modeled using a Beta distribution with shape parameters $sp_{ci} \in \{1, 2, 3, 4, 5\}$. For no more than 70% of all plans verification times are quite close and are thus not shown. From Fig. 6 it can be observed: (1) The larger the skew (sp_{ci}) the higher the verfication time for a small quantile of plans (upper 15%). (2) The larger the skew, the larger the difference between uniformly distributed changes ($sp_{ci} = 1$) and heavily skewed IT changes ($sp_{ci} = 5$) tends to grow for a small percentage of plans. (3) However, in Fig. 6 one measurment for $sp_{ci} = 5$ performs better than the one for $sp_{ci} = 4$ from the 97% quantile of all plans onwards although $sp_{ci} = 5$ means a higher skew. Thus, larger skew of reads / writes on CIs in the CMDB can only be an indicator of higher verification times. We conclude that (1) the higher the skew, the more likely plans take longer to verify, especially if several plans have been processed before as conflicts are more likely to occur. (2) Verification times also depend on the actual CIs addressed as evidenced by the two different measurements for $sp = 5$ yielding different runtimes when reads and writes address different CIs while preserving the skew.

*4) **Number of CIs read and written by a CR**:* The more reads and writes change requests conduct on CIs in the CMDB, the more likely conflicts with other CRs become. CRs that address many CIs have the potential to form large CGs by merging previously independent CGs into a new large one. The unfavorable influence of many reads and writes becomes the stronger the more skewed CRs are over CIs in the CMDB (see Paragraph V-C3) because this emphasizes the likelihood of conflicts. However, an increased number of reads and writes over a set of plans does not necessarily come at higher costs. The CRs might still have their reads and writes distributed in such a way over the CMDB that conflict groups remain small and isolated. We conclude that the more reads and writes CRs conduct, the more likely we are to observe longer verification times due to increased conflict group sizes.

*5) **Total number of CRs / plans processed before**:* Fig. 7 depicts key figures regarding the sizes of CGs as they develop when verifying several sequences of simulated plans and CRs. The following can be observed: (1) The later a plan appears, the larger the size of the biggest CG to verify tends to be (black boxes). (2) The later a plan appears, the more CGs need to be taken into account when verifying a plan compared to previous plans of the same length (white boxes). However, there are exceptions to the rule, for example plan 70 which is very short and thus engages only in a few conflicts. It can be verified in 10ms despite its tail position in the sequence. Due to (1) and (2) we conclude that, the more plans we have processed before the arrival of a new plan the more expensive it is to verify the new plan for conflict freeness.

*6) **Size of the CMDB**:* Larger CMDBs can be beneficial for the performance of the proposed algorithm. If IT changes are equally distributed, e.g., because IT change plans deploy VMs on randomly chosen physical machines, we expect a better performance for the same workload of plans over a large CMDB due to decreased conflict likelihoods. If IT changes exhibit skew over a subset of CIs in the CMDB, e.g., IT changes over the network, we would also expect to see a better performance on a larger CMDB where the number of CRs remains the same. However, if the workload of plans contains IT changes that target exactly the same CI, e.g., due to repeating maintenance changes to a particular CI, the proposed solution does not benefit from larger CMDBs.

D. Performance Measurements

Based on the complexity of the example CRs used in this paper, our implementation is capable of processing 2400 CRs per second when processing the CRs in a topological order (Alg. 2, Lines 4–11). For a CG containing n CRs, $n! * n$ CRs need to be checked in the worst case if no temporal constraints exist among the CRs. This leads to the following approximated worst case verification times of CGs of different sizes: 1.8s for a CG of size 6, 14.7s for a CG of size 7, 134.4s for a CG of size 8. Unordered CGs beyond the size of 8 CRs are not solvable within a few minutes any more. However, it can be expected that conflicting CRs within the same plan are ordered due to causal dependencies among them reducing the number of topological orders to check. Assuming a combination of IT change plans were each IT change depicted in Fig. 3 is part of a distinct plan without additional conflicts involving the given example CRs a CG of size 5 which can be quickly checked.

VI. Conclusions and future work

We identified a research gap when it comes to detect conflicts among IT change plans in multi-operator Change Management environments and by IT change planning solutions [3], [4], [5], [6], [7], [8]. To prevent the occurrence of conflicting IT changes, threatening to render IT change plans infeasible, we proposed an approach for the automated detection of conflicting IT changes. We showed the feasibility of our solution by applying it to several changes from the network and service management domain. Using simulation we discussed the effect that different characteristics of IT changes have on the time needed to decide conflict freeness. Among other criteria we conclude that plans comprising highly skewed CRs and read / write intensive CRs are expensive to verify.

The results presented herein are promising regarding the verification of conflict freeness among practical IT changes within a few minutes as long as no more than 8 CRs are transitively conflicting with each other. It remains to be examined whether change plans in a real environment satisfy this constraint. Furthermore, conflicts destroying the effects of CRs or the assertions of currently running applications have not been addressed in this work. In addition to that, we plan to investigate how semantic knowledge about the effects and preconditions of IT changes can be used to more efficiently reason about conflict absence among IT changes.

References

[1] "IT Infrastructure Library: ITIL Service Transition, version 3. London: The Stantionery Office," 2007.

[2] "Official ITIL Site," online, Jun. 2010. [Online]. Available: http://www.itil-officialsite.com/home/home.asp

[3] A. Keller, J. Hellerstein, J. Wolf, K.-L. Wu, and V. Krishnan, "The CHAMPS System: Change Management with Planning and Scheduling," in *Proc. IEEE/IFIP Network Operations and Management Symp. (NOMS'04)*, Seoul, South Korea, Aug. 2004, pp. 395–408.

[4] K. E. Maghraoui, A. Meghranjani, T. Eilam, M. H. Kalantar, and A. V. Konstantinou, "Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools," in *Proc. CM/IFIP/USENIX Int. Middleware Conf.*, Melbourne, Australia, Dec. 2006, pp. 404–423.

[5] W. L. da Costa Cordeiro, G. S. Machado *et al.*, "A Runtime Constraint-Aware Solution for Automated Refinement of IT Change Plans," in *Proc. IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM'08)*, Samos Island, Greece, Sep. 2008, pp. 69–82.

[6] S. Hagen, N. Edwards, L. Wilcock, J. Kirschnick, and J. Rolia, "One Is Not Enough: A Hybrid Approach for IT Change Planning," in *Proc. IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM'09)*, Venice, Italy, Oct. 2009, pp. 56–70.

[7] D. Trastour, R. Fink, and F. Liu, "ChangeRefinery: Assisted Refinement of High-Level IT Change Requests," in *Proc. IEEE Int. Symp. on Policies for Distributed Systems and Networks (POLICY'09)*, London, UK, Jul. 2009, pp. 68–75.

[8] S. Hagen and A. Kemper, "Model-based Planning for State-related Changes to Infrastructure and Software as a Service Instances in Large Data Centers," in *Proc. IEEE Int. Conf. on Cloud Computing (CLOUD'10)*, Miami, USA, Jul. 2010.

[9] S. Hagen and A. Kemper, "Facing the Unpredictable: Automated Adaption of IT Change Plans for Unpredictable Management Domains," in *Proc. of 6th IEEE/IFIP Int. Conference on Network and Services Management (CNSM'2010)*, Niagara Falls, Canada, Oct. 2010.

[10] A. Keller and S. Subramanian, "Best Practices for Deploying a CMDB in large-scale Environments," in *Proc. of 11th IFIP/IEEE Int. Symposium on Integrated Network Management (IM'2009)*, Long Island, New York, Jun. 2009, pp. 732–745.

[11] DMTF. CIM Standard. [Online]. Available: http://www.dmtf.org/standards/cim/

[12] R. Rebougas, J. Sauve, A. Moura, C. Bartolini, and D. Trastour, "A decision support tool to optimize scheduling of IT changes," in *Proc. of 10th IFIP/IEEE Int. Symposium on Integrated Network Management (IM'2007)*, Munich, Germany, May 2007, pp. 343–352.

[13] L. Zia, Y. Diao, C. Ward, and K. Bhattacharya, "Using Mixed Integer Programming to Schedule IT Change Requests," in *Proc. of 11th IEEE/IFIP Network Operations and Management Symposium (NOMS'2008)*, Salvador, Bahia, Apr. 2008, pp. 895–898.

[14] W. L. da Costa Cordeiro, G. S. Machado *et al.*, "ChangeLedge: Change Design and Planning in Networked Systems based on Reuse of Knowledge and Automation," *Computer Networks: The Int. J. of Computer and Telecommunications Networking*, vol. 53, pp. 2782–2799, 2009.

[15] W. L. da Costa Cordeiro, G. Machado *et al.*, "A Template-based Solution to Support Knowledge Reuse in IT Change Design," in *Proceedings of the Eleventh IFIP/IEEE Network Operations and Management Symp. (NOMS)*, 2008, pp. 355–362.

[16] G. S. Machado, F. F. Daitx *et al.*, "Enabling Rollback Support in IT Change Management Systems," in *Proc. of 11th IEEE/IFIP Network Operations and Management Symposium (NOMS'2008)*, Salvador Bahia, Brazil, Apr. 2008, pp. 347–354.

[17] G. S. Mechado, W. L. da Costa Cordeiro *et al.*, "Refined Failure Remediation for IT Change Management Systems," in *Proc. of 11th IFIP/IEEE Int. Symposium on Integrated Network Management (IM'2009)*, Long Island, New York, Jun. 2009, pp. 638–645.

[18] J. A. Wickboldt, L. A. Bianchin *et al.*, "Improving IT Change Management Processes with Automated Risk Assessment," in *Proc. IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM'09)*, Venice, Italy, Oct. 2009, pp. 71–84.

[19] J. A. Wickboldt, G. S. Machado *et al.*, "A Solution to Support Risk Analysis on IT Change Management," in *Proc. of 11th IFIP/IEEE Int. Symposium on Integrated Network Management (IM'2009)*, Long Island, New York, Jun. 2009, pp. 445–452.

[20] R. C. Lunardi, W. L. da Costa Cordeiro *et al.*, "ChangeAdvisor: A Solution to Support Alignment of IT Change Design with Business Objectives/Constraints," in *Proc. IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM'09)*, Venice, Italy, Oct. 2009, pp. 138–151.

[21] R. C. Lunardi, F. G. Andreis *et al.*, "On Strategies for Planning the Assignment of Human Resources to IT Change Activities," in *Proc. of 12th IEEE/IFIP Network Operations and Management Symposium (NOMS'2010)*, Osaka, Japan, Apr. 2010, pp. 248–255.

[22] D. Koenig, A. Glover *et al.*, *Groovy in Action*. Manning, 2007.

[23] T. Eilam, M. H. Kalantar, A. V. Konstantinou, and G. Pacifici, "Reducing the Complexity of Application Deployment in Large Data Centers," in *Proc. 9th IFIP/IEEE Int. Symp. on Integrated Network Management (IM'05)*, Nice, France, May 2005, pp. 221–234.