Facing the Unpredictable: Automated Adaption of IT Change Plans for Unpredictable Management Domains

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

Alfons Kemper
Technische Universität München (TUM)
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 IT infrastructure and services to satisfy business goals and to minimize costly disruptions on the business. As part of this process, IT changes are planned for, authorized, tested, scheduled, and executed. Unpredictable incidents, e.g., resource failures or unexpected changes in between planning and execution can render plans infeasible. The execution of unsound plans ultimately leads to service disruptions which threaten the continuity of the business. In order to tackle this problem, we propose a logically sound approach to adapt infeasible IT change plans to a changed management domain in order to render them feasible again. We apply our approach to a case study comprising Virtual Area Network (VLAN) configuration and the deployment of a 2-tier service. For the case study we show that the proposed solution can render IT change plans feasible within minimal time differences regarding unpredictable resource conflicts in 95% of all cases if resource utilization remains below 85%.**

I. Introduction

The *Information Technology Infrastructure Library* (ITIL) [1] is a set of best practices that are widely accepted to manage IT services. They comprise the Strategy, Design, Transition, Operation, and Improvement of IT services. In particular *Service Transition* [2] ensures that meeting costs, efficiency, and business needs is achieved with the highest degree of confidence and optimization. *Change Management*, a core ITIL process part of Service Transition, ensures that changes are conducted in a way that risks to the business of a company are minimized. To ensure this, a Change Management process comprising the evaluation, authorization, planning, test, scheduling, implementation, documentation, and review of IT changes is proposed [2].

After IT changes were planned, they are scheduled to be executed in in a change window (a timespan), usually 2-4 weeks after planning. While time passes until plan execution, plans are rendered infeasible due to several reasons: (1) Resources might fail in between planning and execution, i.e., they are unavailable at execution time. However, in previous works [3], [4] we argue that resources need to be already addressed at planning stage because otherwise no logical execution guarantees can be given about the feasibility of the plan. (2) Current state of the art planners for IT changes [3], [4], [5], [6], [7] do not take into account the effects changes being currently planned for have on the feasibility of scheduled, but not yet executed IT changes. Thus, current planners can render previously generated plans infeasible because they do not have any knowledge about them. (3) Some changes might still be conducted manually by operators threatening to invalidate automatically computed change plans. (4) Changes might, although not advised, still be conducted outside of a streamlined Change Management process rendering existing plans infeasible. The adaption of IT change plans to changed management domains is unavoidable because the reliability and business continuity of an organization is at stake by infeasible IT change plans. Such plans can, at best, only be executed partially, leading to inconsistent states and unsatisfied business objectives. Furthermore, according to ITIL [2], a low change success rate is among the top five indicators of poor Change Management. In addition to that, manual change plan adaption is time consuming and subtle logical errors can be introduced that are difficult to detect manually.

To the best of our knowledge, the IT change plan adaption problem has not yet been addressed in previous works on IT Change Management. To tackle the aforementioned problems, we propose an automated, logically sound approach to adapt IT change plans to changed management domains to render infeasible change plans executable. We show the feasibility of our approach by applying it to a network configuration and deployment case study of a 2-tier architecture using virtual appliances [8], [9]. Using our case study we show that our randomized approach scales nicely regarding performance penalties for different types of unpredictable changes, even for large, heavily (85%) loaded data centers in 95% of all cases. The advantages of our approach lie 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 software and hardware of a data center. Fur-

thermore, the solution can be applied to automatically and manually generated IT change plans.
The remainder of this work is organized as follows: In Section II we discuss related work and its shortcomings with respect to unpredictable IT management domains. Section III introduces the basic building blocks of our solution, followed by our Algorithm in Section IV. We evaluate our solution in Section V. Finally, Section VI concludes the work.

II. Related Work

Different aspects of IT Change Management have been addressed in the last recent years. However, despite the negative influence of unpredictable changes on the feasibility of IT change plans - to the best of our knowledge - nobody has yet proposed an approach to adapt plans to changed management domains. Works on IT change plan generation can be roughly divided into two classes: The first class [12], [13], [14] comprises approaches that do not apply logically sound reasoning to IT changes. CHAMPS [12], the seminal work in this class, formalizes planning and scheduling as an optimization problem and achieves a high degree of parallelism. However, Keller et al. [12] do not reason about preconditions and effects of changes. Thus, the generated plans are not guaranteed to be executable from a logical point of view. Similar guarantees are provided by Cordeiro et al. [13], [14] who propose an approach focusing on the reuse of knowledge in IT change design. The authors propose an algorithm to refine abstract IT changes. Compared to CHAMPS, Cordeiro et al. address the reuse of knowledge in IT change design, but still no logical execution guarantees are provided. The second class [3], [4], [5], [6], [7], including our own research [3], [4], comprises algorithms that reason about the preconditions and effects of IT changes. All approaches of this class guarantee the feasibility of the change plan only as long as the management domain remains unchanged. Maghraoui et al. [5] proposed the seminal work in this class. The authors use a *Partial Order Planning* algorithm over a knowledgebase based on predicate logic. The inconvenient transformation of descriptions between object oriented models and predicate logic is the main drawback of this work. Cordeiro et al. [6] propose an algorithm for the refinement of IT changes taking the effects of IT changes on subsequent changes into account as well. Different to [5], their work focuses on the refinement of IT changes. In [3] we proposed an approach to reason about the refinement and the behavior of domain objects interchangeably. Compared to [6], our work made the behavior of domain objects explicit leading to knowledgebases that are easier to adapt and to perceive. Trastour et al. [7] propose a pure refinement based approach. Compared to our work [3], refinement rules are mixed with behavior descriptions making it more difficult to distinguish best practices from behavior. In a subsequent work [4], we propose an approach to reason about state-related changes to Infrastructure- and Software as a Service instances in large data centers preserving its runtime characteristics even for large OO CMDBs and heavily loaded data centers. Compared to our previous work [3], this work focuses on state-related changes and performance issues. Regarding the execution of IT change plans, Machado et al. [15] 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 adapts IT change plans before execution to avoid failures brought about by changed management domains which then would have to be rolled back. Wickboldt et al. [16] 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 plan adaption whereas Wickbold et al. [16] apply risk analysis. Recently, Lunardi et al. [17], [18] proposed different strategies for the alignment of IT change plans to business objectives. Our work complements that work because our solution guarantees the feasibility of IT changes on a logical level for unpredictable domains, a prerequisite to successfully align IT changes with business objectives.
Note, that all works assume predictable management domains and are broken by unpredictable changes. To tackle this problem, Section III introduces the basic building blocks of our solution.

III. IT Change Plan Adaption

A. IT Change Plans for Unpredictable Management Domains

Three different solutions are conceivable to adapt infeasible IT change plans to a changed management domain: *Replanning*, *plan adaption*, and *parameter adaption*. *Replanning* generates a new plan from scratch. If plans are generated manually, an operator has to construct a new plan. This is time consuming and errors are likely to occur. If plans are generated automatically [3], [4], the whole planning process has to be started again. Depending on the planning problem and current state of the CMDB this involves significant effort to create a new plan, i.e., to decide on the IT changes, their parameters, and their precedence constraints. This can involve heavy and costly backtracking through the search space of plans. Compared to replanning, *plan adaption* does not create the plan from scratch, but adapts parts of the plan. It cannot be used together with manually generated change plans because it has to be closely integrated with the algorithm previously used to create the plan. *Parameter adaption* keeps the structure of the original plan, i.e, its changes and precedence constraints, intact and only adapts the plan's parameters, i.e., assigns new values to parameters, to render it executable again. Note, that parameter adaption is less powerful than replanning and plan adaption because it can only render plans feasible again if their structure does not need to be altered. However, parameter adaption has two significant advantages over replanning: (1) Parameter adaption is independent of the planning technique (manual/automated generation of plans) (2) Parameter adaption does not need to go through the process of generating a new plan from scratch. As a first start, this work focuses on a solution using parameter adaption. We plan to investigate replanning, plan adaption, and their trade-offs in respect to parameter adaption in a subsequent work.

B. Importance of Object Oriented Models

Our solution 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 [19] 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]. However, we use our own customized models to keep matters simple. Note, that the approach proposed in this work is independent of the concrete object oriented model used for the CMDB. The CMDB used throughout this work, comprises classes for physical machines (PMs), virtual machines (VMs), operating system (OS) images, web application server (WAS) images, database (DB) images, switches, network interfaces (NIC), and ports. Relationships among them are modeled using references. Effects of IT changes transform a *Configuration Item (CI)*, i.e., an object, of the OO CMDB by writes to its properties to reason about chains of preconditions and effects of subsequent changes in a plan.

C. Parameter Enhanced IT Change Plans

This subsection introduces the concepts our solution is built on. A *change request* (CR) cr is a tuple $cr = (Par_{cr,in}, Par_{cr,out}, Pre_{cr}, e_{cr})$ where $Par_{cr,in} = \{ip_{cr,1}, ..., ip_{cr,n}\}$, $n \in \mathbb{N}$, is a set of input parameters and $Par_{cr,out} = \{op_{cr,1}, ..., op_{cr,n}\}$, $n \in \mathbb{N}_0$, is a set of optional output parameters of cr. Note, that two CRs cr and cr', $cr \neq cr'$, can exist, such that a parameter p exists with $p \in Par_{cr,in} \wedge p \in Par_{cr',in}$, i.e., the same parameter can be input to different CRs. Furthermore, input and output parameters are related such that for every $p \in Par_{cr,out}$ at least one change request cr', $cr \neq cr'$, exists with $p \in Par_{cr',in}$, i.e., output parameters are input parameters to at least another change request. A parameter $par_{cr,d} \in Par_{cr,in/out}$ is a 3-tuple (d, v, V) where d is the description of $par_{cr,d}$, v the current value of $par_{cr,d}$, and V alternative values for v. We denote by $ip_{cr,x}$/$op_{cr,x}$ the input/output parameter x of change request cr. $Pre_{cr} = \{pre_{cr,1}, ..., pre_{cr,n}\}$, $n \in \mathbb{N}$, is a set of preconditions of cr that need to account in order to apply cr. A precondition $pre_{cr,i} \in Pre_{cr}$ accesses a subset of parameters in $Par_{cr,in}$. For example, Listing 1 describes change request *create VM* depicted as cr_1 in Figure 1 using a Groovy [19] *Domain Specific Language*. The precondition in Line 7 checks whether enough memory is available on the physical machine to create a new VM. It accesses the value of input parameter $ip_{cr_1,pm}$ and $ip_{cr_1,vmem}$ which is described as dynamically executable Groovy code over object oriented models. For a change request cr, e_{cr} describes the effects of cr when it is applied. For example, Lines 11-13 in Listing 1 describe the effects of CR *create VM*. A new virtual machine instance is created using the values of the input parameters and is assigned to the value of output parameter $op_{cr_1,vm}$.

Based on the notion of change requests we define our notion of an IT change plan as follows: Let CR be the set of all change requests. A plan pl is a tuple $pl = (CR_{pl}, <_{pl})$ with $CR_{pl} \subseteq CR$ and $<_{pl} \subseteq CR_{pl} \times CR_{pl}$ a partial precedence relation describing Finish-to-Start constraints among the CRs of plan pl. Note, that $(cr_1, cr_2) \in <_{pl}$ iff cr_1 has effects on cr_2 that render cr_2 feasible. $<_{pl}$ is the precedence relation that is given to the scheduler in a subsequent step to schedule the CRs in CR_{pl}. Note, that unordered, i.e., parallel, CRs can exist because the partial order $<_{pl}$ does not need to be total. For a plan pl, $top\_orders(pl)$ is the set of all topological orders of CR_{pl} in respect of the precedence constraints $<_{pl}$.

To assign new values to parameters of an infeasible IT change plan reasoning regarding the dependencies among parameters becomes necessary. For example, the value of some parameters might be determined or influenced by the value of another parameter. To capture these dependencies four different types of parameters are distinguished:

Non-changeable input parameter (NCIP): A Non-changeable input parameter $ip \in Par_{cr,in}$ is a parameter with a fixed value, i.e., $V = \emptyset$. NCIPs are depicted in black, e.g., $ip_{cr_1,vmem}$ or $ip_{cr_1,vcpu}$ in Figure 1. If a precondition of a CR fails due to the value of a NCIP, no alternative value can be assigned to it because this contradicts the intention of the plan. For example, cr_1 in Figure 1 needs to create a VM having exactly the memory described by the value of $ip_{cr_1,vmem}$, not more or less.

Changeable input parameter (CIP): A changeable input parameter $ip \in Par_{cr,in}$ can change its value, i.e., $V \neq \emptyset$, without contradicting the intention of the plan. CIPs, e.g., $ip_{cr_1,pm}$ in Figure 1, are depicted in white. If cr_1 fails because a precondition involving parameter $ip_{cr_1,pm}$ is not fulfilled, another value $v' \in V$, i.e., a new physical machine, can be chosen for the value of parameter $ip_{cr_1,pm}$ to create the VM on another physical machine.

Dependent input parameter (DIP): A dependent input parameter $ip \in Par_{cr,in}$ changes its value with the value of the parameter it depends on. DIPs are depicted in gray. For example, the value of parameter $ip_{cr_3,pmnic}$ depends on the value of parameter $ip_{cr_1,pm}$ because cr_3 configures the network interface of the physical machine used to create the VM in cr_1. If cr_3 cannot be executed, we cannot choose any arbitrary value for $ip_{cr_3,pmnic}$ but only the network interface of the parameter (physical machine) it depends on. DIPs are symbolized by an arrow with white arrowhead starting at the parameter they depend on. Associated with the arrow is a rectangle with round shapes describing how the DIP is derived by traversing the OO CMDB from the value of the parameter it depends on.

Output parameter (OP): To change the value of an output parameter, e.g., $op_{cr_1,vm}$ in Figure 1, at least the value of one input parameter of the same CR needs to be changed. For example, changing the value of $ip_{cr_1,pm}$ leads to a different VM held by $op_{cr_1,vm}$. Note, that this is the only changeable

Listing 1. DSL for change request *create VM*.

```
 1 ChangeRequest(”create VM”, {
 2  NCIP(”vmem”,1024)
 3  NCIP(”vcpu”,2)
 4  CIP(”pm”,list_of_pms)
 5  OP(”vm”)
 6
 7   pre(”enough mem”, {gipv(”pm”).mem_free >= gipv(”vmem”)})
 8   pre(”enough cpu”, {gipv(”pm”).cpu_free >= gipv(”vcpu”)})
 9   pre(”pm running”, {gipv(”pm”).state == ”running”})
10
11   effects {
12    sop(”vm”, new VM(gipv(”pm”),gipv(”vmem”), gipv(”vcpu”)))
13   }
14 })
```

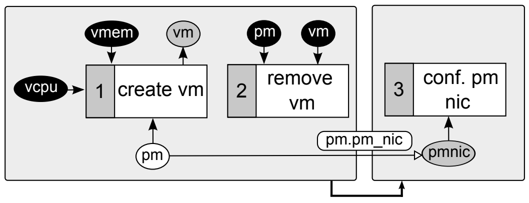

Fig. 1. IT change plan enhanced with parameter relationships.

input parameter of cr_1 because $ip_{cr_1,vmem}$ and $ip_{cr_1,vcpu}$ are both non-changeable input parameters.
The type of a parameter is either defined by an IT practitioner creating a change plan, or automatically by an IT change planner [3] based on the refinement rules applied during planning. A change request can have several parameters of different types depending on the semantics the IT practitioner / planner wants to achieve for the plan. Compared to previous notions of IT change plans [3], [14] our solution makes parameters of IT changes and their relationships explicit. This enables us to find new consistent assignments for parameters to render plans feasible again.

IV. The Algorithm

Algorithm 1 depicts the algorithm to adapt an infeasible IT change plan. The algorithm is called with plan *pl*, the plan to be adapted. First, all topological orders of the CRs of *pl* (CR_{pl}) are generated and an iterator over them is instantiated in Line 2. For the plan depicted in Figure 1, $2! * 1 = 2$ topological orders, $\{< cr_1, cr_2, cr_3 >, < cr_2, cr_1, cr_3 >\}$, do exist. A *while* loop (Lines 3-23) iterates over each topological order (*top_order*) of the plan. A list of executed CRs is initialized with the empty list in Line 5. For the chosen topological order of the plan, the *for* loop (Lines 6-21) iterates over each CR in the topological order. If the precondition of the current CR is satisfied (Line 7), its effects are applied to the OO CMDB in Line 8 and it is added to the list of executed CRs (Line 9). For example, consider cr_1, the first CR in the topological order $< cr_1, cr_2, cr_3 >$ of the plan in Figure 1. The preconditions of cr_1 check whether the physical machine is running (see Line 9, Listing 1) and whether sufficient memory and cpus are available (Lines 7-8, Listing 1). For this purpose, the properties *state*, *mem_free*, and *cpu_free* are read from the physical machine object in the OO CMDB which

Algorithm 1 Parameter adaption algorithm

```
 1: procedure ADAPT_INFEASIBLE_PLAN(Plan pl)
 2:     top_orders_it = new Iterator(top_orders(pl))
 3:     while top_orders_it.hasNext() do
 4:         top_order = top_order_it.getNext()
 5:         executed = []
 6:         for cr ∈ top_order do
 7:             if cr.eval_precondition() then
 8:                 cr.apply_effects()
 9:                 executed.add(cr)
10:             else
11:                 params_fail = cr.get_failed_params()
12:                 to_change = get_changeable_
13:                    input_ancestors(params_fail)
14:                 if set_new_asgnmnt(to_change) then
15:                     top_orders_it.reset()
16:                     break
17:                 else
18:                     return false
19:                 end if
20:             end if
21:         end for
22:         undo_crs(executed)
23:     end while
24: end procedure
```

is currently the value of CIP $ip_{cr_1,pm}$. If the precondition of the CR is false (Line 10, Algorithm 1), the *cr* is queried for failed parameters. For example, assume that precondition *enough mem* (Listing 1, Line 7) of *create VM* is not satisfied. The precondition accesses parameters *pm* and *vmem* (see the calls to the method *gipv* to retrieve the values of the input parameters in Line 7, Listing 1). If this precondition fails, $\{ip_{cr_1,vmem}, ip_{cr_1,pm}\}$ is returned by the call in Line 11, Algorithm 1. Generally, this call returns all parameters of a *cr* that are responsible for its infeasibility. Based on this set, the changeable input ancestors, i.e., CIPs that are ancestors respective the dependency relation on parameters, are derived in Lines 12-13. Parameter $ip_{cr_1,vmem}$ is a NCIP, thus it is not returned. Parameter $ip_{cr_1,pm}$ is a CIP and depends on no other parameters. Thus, only CIP $ip_{cr_1,pm}$ is returned. In general, all CIPs of any CR in the plan that have an effect on parameters that lead to a failed precondition of *cr* are returned in Line 12. Line 14 chooses an alternative assignment for exactly one parameter in the set to_change that has not yet been chosen before. In our example, only parameter $ip_{cr_1,pm}$ can be assigned a new value. Note, that plan optimization can be performed at this step. For example, a new physical machine could be chosen for $ip_{cr_1,pm}$ that minimizes the waste of resources when executing cr_1. If a new value can be assigned (Line 14), all previously checked topological orders need to be checked again because a changed parameter can invalidate the feasibility of previously executed CRs in any topological order. Thus, Line 15 resets the iterator over the topological orders and Line 16 breaks the loop. If the processing of the current topological order is aborted or if it ends regularly, all effects of executed CRs need to be undone in Line 22 because the next topological order to check needs to see the OO CMDB as if the current topological order had not been (partially) executed. In a previous work [4] we showed that the fastest way to restore object oriented CMDBs is to restore the previously backed up properties of objects. Please refer to [4] for a performance analysis regarding different strategies for storing and restoring OO CMDBs. If no new assignment for the parameter is available, *false* is returned in Line 18.

Note, that it is necessary to check all topological orders of a plan. Assume cr_1 in Figure 1 cannot be executed because $ip_{cr_1,pm}$ lacks enough memory. Because $ip_{cr_1,pm}$ is the only changeable input parameter, the algorithm chooses a new physical machine for $ip_{cr_1,pm}$. By coincidence this machine might match $ip_{cr_2,pm}$ and cr_2 might just free enough memory for cr_1 to succeed. In this case, the order cr_2, cr_1 succeeds and the reverse order fails. Note, that the original plan was sound because the preconditions and effects to remove and create VMs on different hosts did not conflict.

For a failure F, we denote by $|F|$ the number of CRs that need to be checked until the iterator over the topological orders of the plan is reset for the last time in Line 15. For a plan pl, the runtime complexity of Algorithm 1 is:

$$O(\underbrace{|F|}_{\text{adapt time}} + \underbrace{|top\_orders(pl)| * |CR_{pl}|}_{\text{verify time}}) \quad (1)$$

The total runtime of Algorithm 1 is determined by two times, the *adapt time* and *verify time*. For a failure F, the adapt time is a measurement as to how fast a new satisfying parameter assignment can be found. It depends on two factors: (1) The amount of invalid choices available for the value of a parameter which causes a CR to fail and (2) whether the failed CR can be detected early or late, i.e., of the position of the CR in the topological order. The verify time comprises the rest of the algorithm's runtime. It consists of $|top\_orders(pl)| * |CR_{pl}|$ executions of the inner *for*-loop. It is failure independent and depends on the length and the amount of topological orders of the plan.

V. Evaluation

Subsection V-A introduces the case study combining service and network management used to evaluate the algorithm. Subsection V-B examines the influence of the type of failure and the *resource utilization* on the adapt time. Finally, Subsection V-C describes the influence of the plan's partial order on the verify time.

A. Plan- and Failure Case study

Figure 2 depicts the plan for a two-tier deployment case study comprising a database and a web application server. The plan consists of 12 CRs. The precedence constraints of the plan are indirectly described by framed gray boxes, groups of parallel executable changes. If all CRs of the previous parallel group have been finished, the CRs of the subsequent group can be executed. The plan is summarized as follows: First, cr_1 and $cr_{1'}$ create VMs for the database and the web application server in parallel. Note, that both CRs define input parameters pm such that $ip_{cr_1,pm} \neq ip_{cr_{1'},pm}$. However, this does not mean that they cannot hold the same physical machine. After that, OS images are bound in the second parallel group (cr_2 and $cr_{2'}$) as well as a DB image (cr_3), and a WAS image ($cr_{3'}$). The third parallel group starts the VMs (cr_4 and $cr_{4'}$). Finally, the last group configures a *Virtual Local Area Network* (VLAN) such that DB and WAS communicate using the same VLAN. cr_5 configures the port of the switch to which the physical machine the VM is running on is connected such that it only accepts VLAN chunks with VLAN ID $ip_{cr_5,vid}$. This is done by modeling $ip_{cr_5,port}$ as a dependable input parameter (DIP) of $ip_{cr_4,pm}$. $cr_{5'}$ is analogous to cr_5. cr_6 and $cr_{6'}$ configure the virtual network device of the created VMs such that they tag Ethernet frames with VLAN ID $ip_{cr_{6/6'},vid}$. For this purpose $ip_{cr_{6/6'},vmnic}$ is modeled as DIP of the formerly created vm, i.e., $op_{cr_{1/1'},vm}$. Seven different types of failures due to unpredictable events occurring after planning are examined:

- **F-MEM:** cr_1 or $cr_{1'}$ fails because not enough memory ($ip_{cr_{1/1'},vmem}$) or CPU resources ($ip_{cr_{1/1'},vcpu}$) are available on the physical machine $ip_{cr_{1/1'},pm}$. To correct F-MEM, a new physical machine has to be chosen for $ip_{cr_{1/1'},pm}$ because all other parameters are NCIPs.
- **F-OS/F-DB/F-WAS:** $cr_{2/2'}$ / cr_3 / $cr_{3'}$ fails because the OS / DB / WAS image defined by $ip_{cr_{2/2'},OSim}$ / $ip_{cr_3,DBim}$ / $ip_{cr_{3'},WASim}$ has already been bound to another VM.
- **F-VLAN:** cr_5 or $cr_{5'}$ fails because the VLAN ID $ip_{cr_{5/5'},vid}$ has already been configured for port $ip_{cr_{5/5'},port}$. Thus, another value for $ip_{cr_{5/5'},vid}$ needs to be chosen to avoid conflicts with applications of other users already using this VLAN ID. Note, that the VLAN ID is input to cr_5, $cr_{5'}$, cr_6, and $cr_{6'}$.
- **F-PORT:** cr_5 or $cr_{5'}$ fails because the port $ip_{cr_{5/5'},port}$ of the switch to which the physical machine $ip_{cr_{1/1'},pm}$ is connected to is deactivated. Thus, a new value for $ip_{cr_1,pm}$ needs to be determined.
- **F-PMNIC:** cr_6 or $cr_{6'}$ fails because the virtual adapter $ip_{cr_{6/6'},vmnic}$ of $ip_{cr_{6/6'},vm}$ cannot be configured because the network interface $ip_{cr_{6/6'},pmnic}$ of the physical machine is deactivated.

B. Influence of Failure and Resource Utilization on Adapt Time

The adapt time depends on the position of the CR causing the failure in the plan and the amount of retries necessary to find a new assignment for invalid parameters. However, for a fixed failure, the amount of retries varies per run because new values for parameters are chosen randomly leading to few retries in case of luck or to much more in case of misfortune.

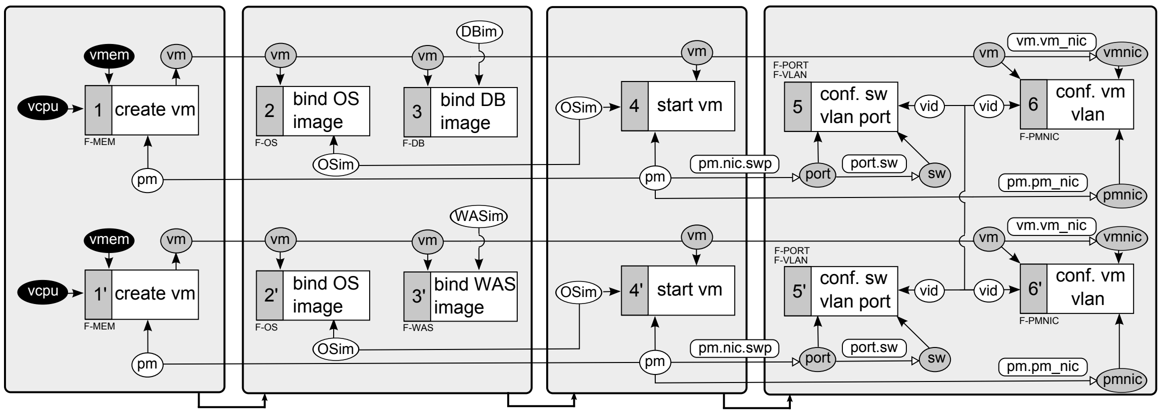

Fig. 2. IT change plan describing the deployment of a 2-tier service and the configuration of a VLAN connection using a switch.

We model the likelihood to choose a value for a parameter that causes a CR to fail as follows: Let cr be an infeasible CR of plan pl due to input parameter ip (not necessarily $ip \in Par_{cr,ip}$). Let N be the total amount of values to choose from for ip, i.e., $N = |V|$, out of which $k < N$ values do not satisfy the failed precondition (blanks). The discrete random variable X denotes the attempt in which a value is chosen for ip that validates the precondition of cr. The *probability mass function* (PMF) $P_{k,N}$ and *cumulative distribution function* (CDF) $F_{k,N}$ of X are described by Equations 2 and 3:
For $i \in \{1, ..., k+1\}, k < N$, N total amount of values to choose from, k number of non-qualifying values / blanks:

$$P_{k,N}(X = i) = \frac{\binom{k}{i-1}(i-1)!\binom{N-k}{1}}{\binom{N}{i}i!} \tag{2}$$

$$F_{k,N}(X = i) = P_{k,N}(X \leq i) = \sum_{j=1,...,i} P_{k,N}(X = j) \tag{3}$$

For a probability p and CDF $F_{k,N}$, the quantile function $Q_{k,N}(F_{k,N})$ in Equation 4 returns the smallest upper bound under which random draws fall in $p * 100\%$ of observations.

$$\begin{aligned}(Q_{k,N}(F_{k,N}))(p) = F_{k,N}^{-1}(p) = \\ = inf\{x \in \{1, ..., k+1\} : F_{k,N}(x) \geq p\}\end{aligned} \tag{4}$$

For example, $Q_{150,200}(0.95) = x$ means that in 95% of all cases no more than x tries are necessary if there are 200 values for a parameter out of which 150 do not satisfy the previously failed precondition. Note, that for several failures in the case study the number of non qualifying values k matches to the resource utilization of the data center. For example, for F-MEM k denotes the amount of PMs heavily loaded, i.e., PMs with not enough RAM or CPU left. For F-OS, F-DB, and F-WAS k denotes the amount of already bound images, and for F-VLAN k describes the amount of already used VLAN IDs. To study the quantile function, and with it the adapt time, depending on the resource utilization of the data center, we define resource utilization as:

$$ru = \frac{k}{N} \tag{5}$$

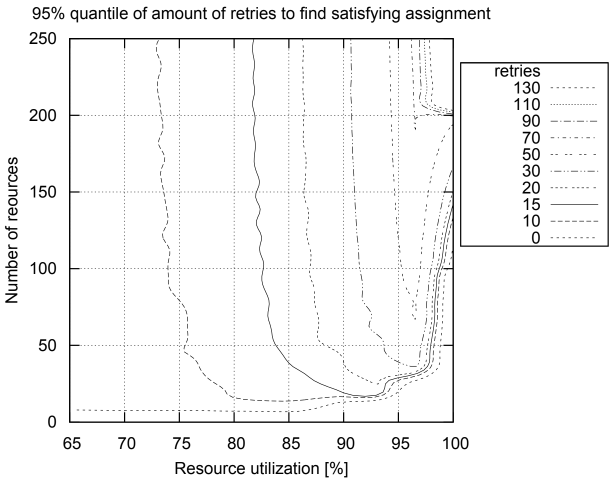

Fig. 3. 95% quantile of the amount of retries necessary to find a new assignement for a parameter depending on the resource utilization and the total amount of CIs qualifying for a parameter.

Figure 3 depicts a contour plot of the 95%-quantile depending on the resource utilization (65%-100%) and the amount of resources, i.e., N (0-250). $Q(0.95)$ is only defined for values above the 0-*contour line* (CL) due to high resource utilization and low amount of total resources below the 0-CL. The following conclusions can be drawn from Figure 3: (1) For 70/80/85% resource utilization no more than 10/15/20 retries are to be expected in 95% of all cases for up to 250 resources in total. (2) The higher the resource utilization, the more retries are necessary. However, the increase is smaller the less resources there are. For example, for 85% resource utilization and 19 resources Q(0.95) holds 12, but for 250 resources it holds 19 (compare different plateus in Figure 3). Thus, (3) for a fixed resource utilization we expect less retries for resources with lesser total amount. However, this increase is not as strong as for increasing resource utilization at constant number of resources. Not shown in Figure 3 are values for lager resource pools of up to 5000 CIs. For such large resource pools the increase is very modest, if at all noticeable, for resource utilizations below 90%. E.g., for 85% resource utilization

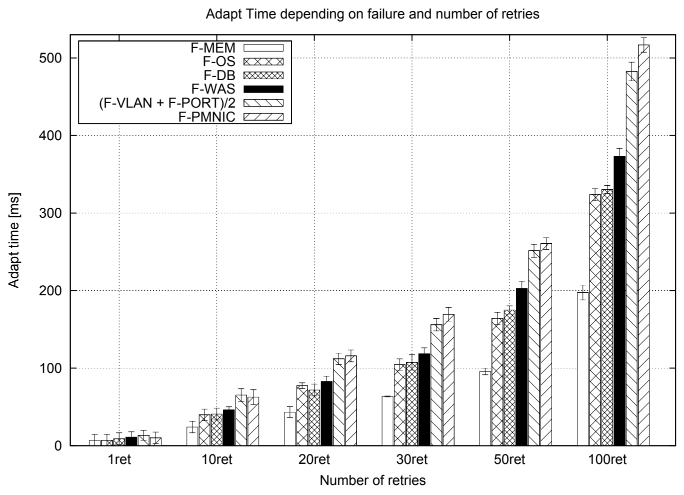

Fig. 4. Adapt time for different types of failures and retries.

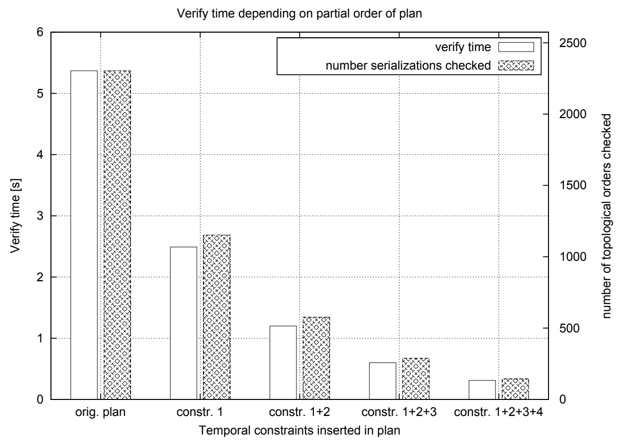

Fig. 5. Verify time depending on additional precedence constraints inserted into the plan. The closer to a total order the faster the verify time. Constr. 1 = (cr_2,$cr_{2'}$), Constr. 2 = (cr_3,$cr_{3'}$), Constr. 3 = (cr_4,$cr_{4'}$), Constr. 4 = (cr_6,$cr_{6'}$).

$Q(0.95)$ holds 19 for 5000 objects, the same value as for 250 objects. Thus, for the presented case study our approach scales well with high amounts of resources under resource utilizations of up to 85% regarding the maximal number of retries (in 95% of all cases). To connect the amount of retries to a runtime, Figure 4 shows the adapt time measured for the different failures in milliseconds depending on the amount of retries. Note, that F-VLAN and F-PORT were consolidated because they are both caused by the same CRs showing similar runtime. All measurements were taken on a Core2 Duo 2.8Ghz, 6MB of Cache, and 4GB of RAM. The underlying OO CMDB comprises 1000 PMs, 3000 OS images, and each 1500 DB and WAS images. From Figure 4 we observe: (1) The adapt time increases proportional to the number of retries for every failure. (2) Failures happening late in the plan are more expansive than early failures (e.g. F-PMNIC vs. F-MEM) at the same number of retries. This is because they can only be detected at a later time when executing the CRs of a topological order. (3) The more retries there are, the larger the gap between early and late failures. This is due to (2) and the increased number of retries which emphasizes the effect from (2). Thus, we conclude that late failures with a high resource utilization regarding the values to choose from are the most expensive ones. However, for a resource utilization below 85% no more than 20 retries occur in 95% of all cases (see quantile plot in Figure 3). From Figure 4 we can conclude that this can involve a worst case performance penalty of around 100ms between early and late failures. For 70% resource utilization ($\leq$ 10 retries) a maximum difference of 65ms can be observed between early and late failures in 95% of all cases. Compared to the verify time explored in Subsection V-C this difference is very small. Higher resource utilizations are very unlikely to occur because this means that a data center operates very close to its capacity limit making it inflexible to cope with varying resource demands. Note, that in the worst case more retries are necessary than the 95% quantile. All in all, for the given case study and for resource utilizations of up to 85% we can consider the plan adaption time to be roughly independent of the failure in 95% of all cases, especially if the length of the verify time examined in Subsection V-C is taken into account.

C. *Influence of Plan on Verify Time*

The verify time depends on the length of the plan ($|CR_{pl}|$) and $|top\_orders(pl)|$, the number of its topological orders. The plan depicted in Figure 2 has $2! * 4! * 2! * 4! = 2304$ topological orders. Thus, the precondition and effects of $12 * 2304 = 27648$ CRs need to be checked / executed after the plan was adapted to make sure that the adaption is sound (Lines 6-21). Note, that all possible topological orders need to be executed because otherwise it cannot be guaranteed that the adaption is sound for non-ordered, i.e., parallel, CRs. Verifying our case study plan takes roughly 5.37s.
However, if additional knowledge about the semantics of the preconditions and effects of CRs is available, $|top\_orders(pl)|$ can be reduced. For example, consider changes cr_4 and $cr_{4'}$ which start the previously created virtual machines. It is not necessary to take the execution order among cr_4 and $cr_{4'}$ into account, because whenever one execution sequence of the two CRs in a topological order of the plan succeeds, the other one will as well. This is due to the preconditions and effects of $cr_{4/4'}$. They have the following preconditions: (1) The two (distinct) virtual machines $ip_{cr_{4/4'},vm}$ are not yet running (2) Both OS images ($ip_{cr_{4/4'},OSIm}$) are connected to their VM ($ip_{cr_{4/4'},vm}$) (3) The physical machines $ip_{cr_{4/4'},pm}$ are on. cr_4 and $cr_{4'}$ have only one effect on the CMDB, they change the state of the VM they start ($ip_{cr_{4/4'},vm}$) to *running*. Looking sharply at preconditions and effects, it can be observed that whenever one execution sequence of the two changes succeeds the other one will as well because their preconditions and effects do not overlap and they do not influence a property of a common CI. Otherwise, their execution order might matter. Thus, it suffices to either check order $cr_4, cr_{4'}$ or $cr_{4'}, cr_4$ but not both. This reduces $|top\_orders(pl)|$ by the factor of two. Other conflict free pairs are for example (cr_2,$cr_{2'}$),

(cr_3,$cr_{3'}$), and (cr_6,$cr_{6'}$). It suffices to verify the original plan (Figure 2) with any combination of these constraints because CRs addressed by the temporal constraints do not overlap in preconditions and effects. Figure 5 depicts the influence of the additional constraints on the verify time (left vertical axis) and the number of topological orders checked (right vertical axis). Each constraint added to the plan reduces the space of topological orders by the factor of two. This approximately halves the verify time for every constraint added. While the original plan takes 5.37s and 2304 topological orders to verify, the plan with the aforementioned 4 constraints has only 144 topological orders and takes about 0.31s. Note, that for a given plan the verify time is constant (except for variances in measurements) leading to the fact that the adapt time (see Section V-B) determines the total runtime dependent on the type of resource conflicts. To efficiently apply our approach to larger and highly parallel plans, finer grained reasoning techniques about preconditions and effects of IT changes become necessary because in the worst case (no precedence constraints among CRs of a plan) the amount of topological orders of a plan can grow factorially with the amount of CRs it contains. However, in the best case it remains constant if the plan is a sequence of IT changes. In the context of this work, we rely on an operator to describe conflict free CRs that are unaffected by parameter changes to successfully reduce the amount of topological orders to be checked. We are currently developing finer grained reasoning techniques over object oriented models as part of an algorithm to automatically detect conflicting IT change plans.

VI. Conclusions and future work

Change plan generation is a crucial step of the IT Change Management process part of ITIL. In between the generation and execution of IT change plans unpredictable changes can render IT change plans infeasible. This increases the quantity of failed IT change plans, an indicator of poor Change Management [2]. Existing automated solutions do not take into account this gap that can alter change plans infeasible. To tackle this problem, we proposed an automated, logically sound approach to adapt IT change plans to changed management domains using parameter adaption. We examined the times to correct several different types of resource conflicts in a deployment and network configuration case study. For our case study we found that our approach is able to find a solution within minimal time differences for different types of unpredictable resource conflicts/failures in 95% of all cases, as long as resource utilization remains below 85%.

For future work, we envision an automated approach to automatically detect and correct conflicting IT change plans over OO CMDBs. As part of this vision we develop techniques to reason about preconditions and effects of IT changes over OO models. Furthermore, replanning and plan adaption techniques, including trade-offs, remain to be investigated.

References

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

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

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

[4] 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. of IEEE Int. Conf. on Cloud Computing (CLOUD'10)*, Miami, USA, Jul. 2010, pp. 11–18.

[5] 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. of CM/IFIP/USENIX International Middleware Conference*, Melbourne, Australia, Dec. 2006, pp. 404–423.

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

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

[8] C. Sun, L. He, Q. Wang, and R. Willenborg, "Simplifying Service Deployment with Virtual Appliances," in *Proc. of IEEE Int. Conf. on Services Computing (SCC'08)*, Honolulu, HI, Jul. 2008, pp. 265–272.

[9] A. V. Konstantinou, T. Eilam *et al.*, "An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds," in *Proc. of the Int. Workshop on Virtualization Technologies in Distributed Computing (VTDC'09)*, Barcelona, Spain, Jun. 2009, pp. 9–18.

[10] A. Keller and S. Subramanian, "Best Practices for Deploying a CMDB in large-scale Environments," in *Proc. of IEEE/IFIP Int. Symposium on Integrated Network Management (IM'09)*, Long Island, New York, Jun. 2009, pp. 732–745.

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

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

[13] 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.

[14] W. L. da Costa Cordeiro, G. Machado *et al.*, "A Template-based Solution to Support Knowledge Reuse in IT Change Design," in *Proceedings of IEEE/IFIP Network Operations and Management Symp. (NOMS'08)*, 2008, pp. 355–362.

[15] G. S. Machado, F. F. Daitx *et al.*, "Enabling Rollback Support in IT Change Management Systems," in *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS'08)*, Salvador Bahia, Brazil, Apr. 2008, pp. 347–354.

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

[17] 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. of IEEE/IFIP Int. Workshop on Distributed Systems: Operations and Management (DSOM'09)*, Venice, Italy, Oct. 2009, pp. 138–151.

[18] R. C. Lunardi, F. G. Andreis *et al.*, "On Strategies for Planning the Assignment of Human Resources to IT Change Activities," in *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS'10)*, Osaka, Japan, Apr. 2010, pp. 248–255.

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