Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools

Kaoutar El Maghraoui[1], Alok Meghranjani[2], Tamar Eilam[3], Michael Kalantar[3], and Alexander V. Konstantinou[3]

[1] Dept. of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, USA
[2] École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
[3] IBM T.J. Watson Research Center, Hawthorne, NY, USA

Abstract. Today's enterprise data centers support thousands of mission-critical business applications composed of multiple distributed heterogeneous components. Application components exhibit complex dependencies on the configuration of multiple data center network, middleware, and related application resources. Applications are also associated with extended life-cycles, migrating from development to testing, staging and production environments, with frequent roll-backs. Maintaining end-to-end data center operational integrity and quality requires careful planning of (1) application deployment design, (2) resource selection, (3) provisioning operation selection, parameterization and ordering, and (4) provisioning operation execution. Current data center management products are focused on workflow-based automation of the deployment processes. Workflows are of limited value because they hard-code many aspects of the process, and are thus sensitive to topology changes. An emerging and promising class of model-based tools is providing new methods for designing detailed deployment topologies based on a set of requirements and constraints. In this paper we describe an approach to bridging the gap between generated "desired state" models and the elemental procedural provisioning operations supported by data center resources. In our approach, we represent the current and desired state of the data center using object models. We use AI planning to automatically generate workflows that bring the data center from its current state to the desired state. We discuss our optimizations to Partial Order Planning algorithms for the provisioning domain. We validated our approach by developing and integrating a prototype with a state of the art provisioning product. We also present initial results of a performance study.

1 Introduction

Today's enterprises are increasingly reliant on network-based services to implement mission-critical business processes. A typical enterprise supports thousands of business applications, composed of numerous heterogeneous distributed components, and deployed in multiple large data centers. The collection of business

applications is in a constant state of flux. New applications are developed, tested on different test environments, staged and rolled into production. Existing applications are continuously updated, and often rolled-back. Data center operations personnel must provision and configure multiple networked environments and deploy applications into them. The capabilities of these environments have also evolved from basic services such as routing, naming, and hosting, to higher-level middleware services such as directory, messaging, and storage. Maintaining data center operational integrity and quality has thus become an extremely challenging task. In the absence of end-to-end operational methodologies and tools, enterprises and their customers are exposed to significant operational cost and risk.

While operators enjoy a large set of tools to perform local configuration tasks, they face major challenges in deploying complete applications. Current methodologies and tools provide fragmented and incomplete support for the end-to-end application deployment process. This process can be broken into four major logical steps, representing different domains of expertise. First, a deployment solution must be designed that satisfies functional application deployment requirements and non-functional deployment goals. Second, resources must be selected that can be used to implement the solution. Resource selection must take data center capabilities and constraints into account. Third, an ordering of bound provisioning operations must be established to bring the data center from its current to the desired state. Complex constraints exist between configuration parameters across the various tools and in provisioning operations that are far apart in the ordering sequence [1]. Implied or poorly documented ordering interdependencies are typically discovered in the process of deploying an application. Forth, the selected operations must be invoked across different management platforms and domains. Operation execution status must be monitored, and operational errors reported.

A new class of cross-platform management products, such as IBM Tivoli Provisioning Manager (TPM)[2], has emerged to address the bottom-up challenges of operating heterogeneous data centers. These provisioning technologies offer a large set of automation packages that expose a uniform data access and update layer to heterogeneous management platforms. In addition, they provide a platform for programming workflows for higher-level provisioning operations. Users can create workflows invoking primitive provisioning operations or other workflows to automate the deployment of a business application. Typically, such workflows statically encode significant aspects of the application design, resource selection, and operational ordering choices tied to a particular deployment environment. Development, testing and maintenance of such workflows in a changing environment is a significant challenge. Since these workflows are specific to a deployed application and a target environment, the potential level of reuse is minimal. Thus, the amortized complexity is not reduced.

More recently, a new class of model-based tools has emerged to address the top-down challenges of designing and binding applications to data center resources. In these tools, the current data center state and the desired deployment

solution are both described declaratively using object-relationship models. These tools aid in the construction and validation of a model describing the desired solution, which may vary in its degree of concreteness. In particular, resource instances may or may not be identified by it. In [3, 4] we described the design and implementation of such a tool, and the methods we used to guarantee that the deployment topology satisfies an input set of requirements and constraints. While these tools address the design and resource selection challenges, their output is a declarative model, and the generalized task of realizing this model using existing provisioning tools is an open problem.

We believe that representing configuration knowledge in object models will offer significant advantages and will be the basis for the next generation of configuration management tools [5]. Models provide easier visualization, conceptualization, extensibility, componentization, standardization, and reuse. Most technologies in this space are moving towards model-based solutions. However, there exists an inherent mismatch between a declarative model that is the basis for these modeling tools and the provisioning technologies that are in essence procedural. Moreover, the granularity does not match: models are typically fine grained, while provisioning operations are coarser grained. In particular, provisioning operations may have a complex effect on multiple resources. In modeling terms, multiple objects and relationships may change in a model describing the state of the data center before and after an execution of a provisioning operation.

In this paper we describe an approach which bridges the gap between the declarative model of a solution and the procedural provisioning operation tooling needed for its implementation. We use models to declaratively describe the required solution, as well as the operational capabilities of existing provisioning platforms. We then employ a planning algorithm to automatically infer the partial order of provisioning operations and their inputs to deploy a given application in a data center. The generated workflows maintain operational constraints while verifiably provisioning the desired data center state. Our approach supports the seamless integration of existing automation tools specializing in application solution design, resource selection, and cross-platform provisioning. We based and evaluated our models and algorithms on the capabilities of a state of the art provisioning product[2], in customer use.

The structure of the paper is as follows. In Section 2, we present an overview of our approach and architecture for model driven deployment planning. Next, in Section 3, we present a formal model for applying planning to the problem domain. In Section 4, we describe the planning algorithm and how we optimized it for our particular usage of deployment planning. In Section 5, we describe our prototypical implementation and integration with TPM [2], a state of the art provisioning product. In Section 6, we present empirical results of a multi-tier network provisioning experiment. Last, in Section 7, we review the related work, and in Section 8, we summarize the work and discuss future challenges.

2 Approach

2.1 Background: Data Center Operations Today

Data center operators use a large number of automation and configuration tools to deploy distributed applications and services on a set of managed resources. Every such tool provides a set of automated resource management functions, that we term *provisioning operations*. To accomplish a particular configuration task, the operator must identify a set of provisioning operations provided by the collection of available tools, instantiate them correctly, and execute them in an appropriate order. Hundreds of low-level provisioning operations may be required to deploy a single application. For example, servers must be selected from a free pool, network switches, firewalls and load balancers must be configured, operating systems, middleware, and application components must be installed and/or configured, and monitoring must be enabled. Some of these tasks, such as the selection of resources, are performed manually, others with tooling assistance. The ordering and parameterization of provisioning operation invocations is determined by operators in an ad hoc manner. Operators typically rely on past experience, product manuals, existing scripts and other unstructured and informal data sources.

Individual provisioning operations may incorporate complex logic. A provisioning operation often makes assumptions about the state of the affected resources and about other resources connected to them. Upon invocation, it may perform a large number of fine grained configuration actions effecting the state of a number of resources in the data center. For example, a provisioning operation to install an operating system may need to configure a DHCP server and a network image server, in addition to the target system of the installation. In secure environments, the OS install operation might have to configure a number of network devices to ensure connectivity between the install server and the target server. To determine a successful order of executions, operators must fully understand the preconditions and effects of each of the provisioning operations and their interdependencies. Due to the aforementioned complexities, operators often rely on step-by-step trial and error operation. Even a simple application migration from the developer's workstation to a testing environment can become a challenge, with studies indicating it accounts for 35% of the testing time[4].

2.2 A Case for Model Driven Deployment Using Planning

Increasingly, object models are used in order to formally describe resource configuration state. The objects in these models are typically typed and associated with attributes. For example, the Management Information Base (MIB) of an IP system will contain an object for each IP network interface, with attributes such as IP address and netmask. The IP interface node will also have a relationship to the network interface card (NIC) object on which it is defined. These configuration models can be navigated and queried at a fine level of granularity.

[4] Theresa Lanowitz, speaking at a Mercury Users Conference, 2004

While configuration models offer uniform access and navigation, their update functions are typically not uniform or consistent. A single operation may have multiple parameters, pre-conditions and post-conditions. For example, the operation to configure an IP interface may take multiple parameters such as the IP address, netmask, NIC, and default gateway, where the IP address must be unique and must match the netmask, the NIC must be enabled and connected to a link-layer network associated with the netmask, and so on. The execution of the operation may result in changes at multiple attributes and nodes in the MIB.

The complexity of configuration operation parameterization and dependency ordering, necessitates the use of advanced algorithms for inferring and generating correct sequences of provisioning actions, termed *workflows*. The workflow generation problem can be naturally reduced to the AI planning problem. A planning system synthesizes a course of actions to change the world from its current state to a desired goal state. A planning domain defines a set of atomic *actions* that are capable of changing the state of the world. Each action can only be executed under some particular conditions of the world termed *preconditions*. Each action has certain *effects* on the state of the world. A *planner* generates a *plan*: a sequence of actions that will bring the world from its initial state to the goal state.

Use of planning for workflow generation allows users to focus on the declarative expression of data center resources, desired state and operation models. Models can be defined by different users, supporting separation of concerns across resource types and operational domains. For example, a CISCO router expert may model the pre-conditions and post-conditions of a CISCO IOS router configuration operation. A deployment expert may add a constraint that a route must exist between the boot server and a system being rebooted. These models can be created once, and reused many times to automatically generate multiple workflows to deploy multiple applications in different networked environments. Thus, the amortized complexity of managing the enterprise data center is significantly reduced using this approach.

Our approach to deployment planning and execution can be summarized as follows. (1) we formally capture both the current state of the data center, and the desired deployment solution using object-relationship models, (2) we formally capture the pre-conditions and effects of a key set of provisioning operations provided by the available tools using propositional logic, (3) we employ partial order planning algorithms to automatically generate sequences of provisioning operations to bring the data center from its current state to the desired state. We optimize partial order planning to the area of provisioning by utilizing domain knowledge and data center characteristics.

The detailed architecture of our approach is depicted in Fig. 1. The current state of data center resources is maintained in a configuration database in the form of an object-relationship model. Automation tools are integrated into the system by a specialized adapter. This adapter provides an abstraction layer and a common access layer to execute the functions that are provided by the tool.

In addition, the pre-conditions and effects of key provisioning operations are formally modeled as predicates and transformers over the data center state and kept in a repository.

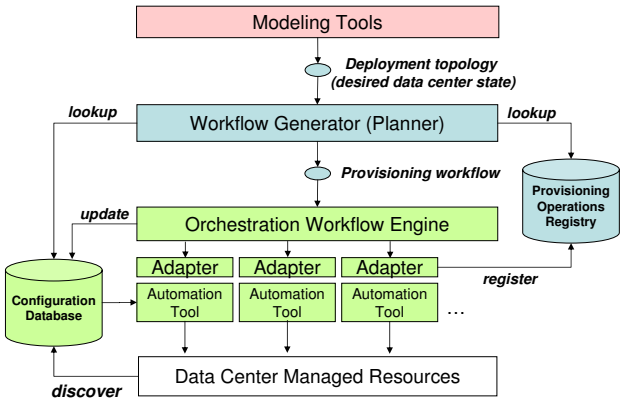

Fig. 1. Architecture

A workflow orchestration framework provides the means to author workflows, encoding sequences of invocations of provisioning operations, and to manage their execution. These worflows, in addition to invoking operations on the tools, update the configuration database, based on the expected or actual result of the execution. This is necessary since the tools are agnostic of the configuration database. Note that this will always be an approximation of the actual state of the data center. Discovery techniques can be employed to fix any inaccuracies in the state of the data center as it is recorded in the configuration database.

A workflow generator (planner) receives three inputs: the current state of the data center, the desired deployment topology, and the available provisioning operations. The desired topology is generated using a modeling tool, such as [4]. The workflow generator employs a planning algorithm to automatically generate orchestration workflows. The workflow generator component is the focus of this paper. The rest of paper focuses on the design, implementation, algorithms, and empirical studies of this architectural component.

3 Generating Workflows Using Planning

Given a model describing the desired deployment state of an application, the task of workflow generation involves identifying the provisioning operations to be performed, binding of operational parameters, and analyzing ordering dependencies between operations. The set of available provisioning operations is determined by the provisioning technology. In this section, we describe how we

can use planning methods to generate the workflows by mapping the models representing current and desired state to first order logic that is the input to most planners, and modeling the provisioning operations as planner actions. Sect. 4 will focus on the planning algorithm and our adjustments and optimizations necessary for it to work well in this domain.

3.1 Problem Domain Modeling

State Modeling Any resource object model (in fact any network model [6]) can be simply mapped to first-order logic. For a given object model, consisting of typed nodes and relationships containing attributes, the following construction will generate an equivalent planning initial or goal state:

- For each object instance N of type T, add the following predicates:
  - `(exists` N`)`, to express that the object is in the *created* state.
  - `(`T N`)`, to express the type of the object.
  - For object models with support for inheritance, add `(`T_1 N`)`, `(`T_2 N`)`, ... predicates for each supertype T_i of type T.
- For each relationship instance E of type T, between N_1 and N_2, add the following predicates:
  - `(established` E N_1 N_2`)`, to express that the relationship is in the *established* state.
  - `(`T E`)`, to express the type of the relationship.
  - For object models with support for relationship type inheritance, add `(`T_1 E`)`, `(`T_2 E`)`, ... predicates for each supertype T_i of relationship type T.
- For each object or relationship O, and an attribute A declared in type T with value V, add the following predicates:
  - `(set` O A`)`, to express that the attribute is set.
  - `(`T`.`A O V`)`, to express the attribute's value.

The above rules are used to translate both the initial data center model and that of a desired topology (which represents the desired state of all the resources in the data center) into a first-order representation.

Figure 2 shows a sample object-relationship configuration model of a data center server and its logic representation. The server contains a network interface card (NIC) which is connected to a switch port on switch `sw01`. The switch port is also configured to be a member of the virtual LAN (VLAN) `vlan1` defined on `sw01`. The `SwitchPort` type is inherited in our type system from the `Nic` type. `vlan1` has an attribute `vlanNumber` with value `201`.

Action Modeling Provisioning operations are modeled as planner actions. Typically, provisioning actions are implemented imperatively, thereby requiring additional declarative modeling of pre-conditions and post-conditions. The Planning Domain Definition Language (PDDL) [7] is a common language for

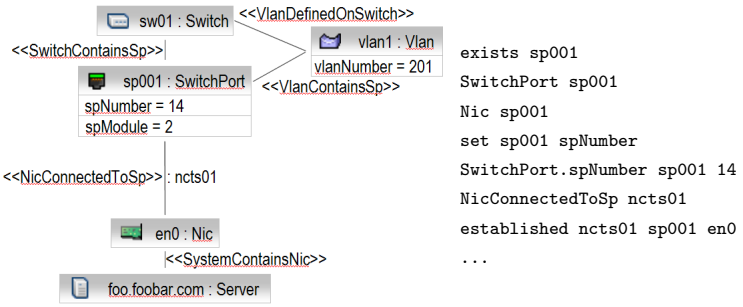

Fig. 2. Example object model and logical representation.

expressing planning domains. Preconditions for each action may express restrictions on the life-cycle state of entities (e.g. exists), graph structure, and attribute values. The effects of the actions are similarly expressed.

An example of a configuration operation model expressed in PDDL is shown in Fig. 3. The operation `moveSwitchPortToVlan` configures a switch to assign a switch port into a particular VLAN. This allows the computer system connected to the port to communicate with other computer systems in that VLAN as if they were on the same local network. In order to move a switch port into a VLAN, the switch port, the switch and the VLAN must all exist and be interconnected. The preconditions clause in Fig. 3 express these requirements. The effects clause indicates that after the execution of this action, the switch port will be contained in the VLAN. For simplicity, we omitted the type predicates for both objects and relationships. They are implied from the definition of the parameters.

```
(:action moveSwitchPortToVlan
    : parameters      (?switch - Switch
                       ?sp     - SwitchPort
                       ?vlan1  - Vlan
                       ?vlan2  - Vlan
                       ?scs    - SwitchContainsSp
                       ?vdos   - VlanDefinedOnSwitch
                       ?vcs1   - VlanContainsSp
                       ?vcs2   - VlanContainsSp)
    : precondition (and (exists ?switch)  (exists ?sp) (exists ?vlan2)
                        (established ?scs ?switch ?sp)
                        (established ?vdos ?vlan2 ?switch)
                        (established ?vcs1 ?vlan1 ?sp)
                        (set SwitchPort.spNumber ?sp) (set SwitchPort.spModule ?sp)
                        (set Vlan.vlanNumber ?vlan2))
    :effect        (and (established ?vcs2 ?vlan2 ?sp)
                        (not (established ?vcs1 ?vlan1 ?sp))
```

Fig. 3. PDDL specification for the `moveSwitchPortToVlan operation`.

3.2 Planning

Planning algorithms are a class of search algorithms. They basically search and backtrack various possible plans until they find a solution. Classic planners adopt one of two approaches: searching the world state space or searching the plan space. In the first approach, the search space consists of a graph whose nodes represent the state of the world and whose edges represent the execution of actions (e.g. GraphPlan [8]). The planner can search world space starting from the initial state (progression planners) or starting from the goal state (regression planners). In the second approach, each node in the graph represents a partial plan, and each edge represents a plan refinement operation. Of these, some algorithms generate totally ordered plans (Total Order Planning), while others generate partially ordered plans (Partial Order Planning or Least Commitment Planning) [9, 10].

For the task of provisioning workflow generation, we selected partial order planning (POP) for the following reasons:

1. A partially ordered plan can be efficiently executed in parallel. Provisioning a distributed application typically requires configuring multiple systems. These configurations can usually be done in parallel.
2. In any given data center state, there may be many possible actions that can be executed. The number of such actions is proportional to the size of the data center. Consequently, world state search approaches will have a high branching factor. A high branching factor is likely to increase planning cost.
3. Partially ordered planning algorithms are efficient when the number of possible actions to fulfill a given condition is small. This is the common case for distributed application provisioning: typically only a few provisioning operations will produce a particular configuration of a given resource.

3.3 Partial Order Planning

A partial order planner searches the plan space. Each node in the search space represents a *partial plan* while edges represent *plan refinement* operations. We briefly review partial order planning; for a more detailed description see, for example, [9].

A partial plan in POP is a set of action steps $S_0, S_1, \ldots, S_f$ and a set of ordering constraints $S_i < S_j$ which indicate the causal order of the action steps. In addition, the following meta information is maintained: (1) a set of causal links $S_i \rightarrow^c S_j$ that record that precondition c of step S_j is achieved by step S_i, (2) a set of *open conditions* which consist of action preconditions that still remained to be achieved, and (3) a set of *unsafe links* $Si \rightarrow_c S_j$ indicating that precondition c is deleted by some step S_k in the partial plan.

Partial order planning begins with an initial unfinished plan comprising two dummy steps: Start Step S_0 and Finish step S_f. S_0 is a step with no preconditions and whose effects represent the world in the initial state. S_f is a step with no effects and whose preconditions represent the world in the goal state. This initial

plan is iteratively refined by applying plan refinement operations also termed *flaw resolutions*. Flaw resolutions fall into two categories:

1. **Open Condition Achievement**: An open condition represents a unachieved precondition of an action already added to the partial plan. An open condition can be achieved by adding a new action to the partial plan or by reusing an action already in the plan.
2. **Unsafe Links Resolution**: An unsafe link indicates that an achieved condition may be invalidated by another action. In this case, ordering the actions avoids the conflict.

A partial plan is a complete plan when there are no flaws in it. A partial planner continues to refine the different partial plans generated until a complete plan is found or all the different possibilities have been tried and no solution is found. Observe that a POP planner generates a plan backwards by identifying actions that achieve the preconditions of actions already in the plan.

General-purpose planning algorithms suffer from poor scalability. The use of domain-specific knowledge is critical to developing practical planners [11]. We show in subsequent sections, how we have exploited the nature of the domain of network configurations in distributed application provisioning to achieve significant improvements in the efficiency of POP for this purpose.

4 Optimizing POP for Provisioning

4.1 Domain Characteristics

The domain of distributed application deployment poses efficiency challenges to the generic partial order planning algorithm:

Complex Provisioning Operations Provisioning operations tend to be complex, ofter performing multiple configuration tasks. Consequently, even simple operations frequently have several parameters and effects. Their preconditions also tend to have many clauses. When adding an action to a partial plan, each parameter that must be instantiated acts as a multiplier to the number of possible variable instantiations, resulting in a high branching factor.

Data Center Size A typical data center manages hundreds, if not thousands, of resources. This means that for a given action parameter, the planner needs to consider a large number of possible values. For example, when trying to instantiate a server variable in an action, there may be hundreds of possible instantiations. In partial order planning, if we naively attempt to fully instantiate each action that is added to a partial plan, the branching factor will be prohibitive.

Constrained Resource Modifications Resources can be configured only in a limited number of ways, and data center policy typically introduces even more configuration constraints. For example, the set of NICs that a server contains is typically fixed. In addition, once a server is wired into a data center, its relationship (through its NIC) to a switch port on a particular switch is typically fixed. Consequently, a resource typically has a number of fixed relationships with other resources. Many provisioning operations tend to be local in nature: they operate on groups of closely related parameters. In subsequent sections we explore how a planner can take advantage of the fixed relationships between resources to efficiently instantiate parameters.

Runtime Parameter Determination Not all parameters for provisioning operations are available at planning time. Some are only available at deployment time. For example, to enable communication, it is necessary to configure a server's network interface with an IP address. It is usually not possible to select any unused IP address; the selection is constrained by a runtime data center policy. As a consequence, at planning time, it is not possible to instantiate all variables. We discuss how we deal with this, using a concept that we term *deferred instantiation*, in Sect. 4.2 below.

4.2 Partial Order Planning for Provisioning

Prioritized Flaw Selection Partial order planning proceeds by iteratively selecting flaws to resolve. This selection can be ordered to improve planning efficiency. In our workflow generator, we, as in [12] and [13], resolve unsafe links before open conditions. With regards to resolving open conditions, we use the following priority (in a descending order):

1. Open conditions of the form `(exists ?<type>)`.
2. Fully instantiated open conditions.
3. Other partially instantiated open conditions.

Our highest priority is to instantiate unknown resources. We do so for two reasons: first, such open conditions have only one uninstantiated variable, helping to reduce branching. Second, once resources are bound, it is more likely that open conditions representing resource relationships and attribute values will be more constrained, again reducing branching.

Fully instantiated open conditions are given preference compared to partially instantiated open conditions because they constrain the problem more. They are least likely to introduce the uninstantiated open conditions.

When comparing two open conditions which are either fully instantiated or which are both partially instantiated, the flaw selection algorithm counts the number of partial plans that will be created to resolve each open condition. The open condition that generates the fewest new partial plans will be selected.

Condition Driven Variable Instantiation Recall that one class of flaw resolutions are open condition achievement operations. These operations involve adding a new action to a partial plan or reusing an existing action in a partial plan to achieve the precondition of another action in the partial plan. When adding actions we can choose to instantiate all of its parameters with specific values or instantiate only some of them. If we choose to instantiate all of the parameters, we create a new partial plan for each combination of fully instantiated variables. Recall that provisioning operations typically have a large number of parameters and that data centers manage a large number of resources. Consequently, a large number of new partial plans will be generated. In search terms, the branching factor will be high. On the other hand, we may leave parameters uninstantiated until a consistency threat necessitates instantiation. If we leave parameters uninstantiated, the branching factor remains low, however, planning becomes more complex, as it is necessary to maintain variable binding constraints that specify whether two parameters are the same. Further, it is necessary to implement a unifier that takes into account the variable binding constraints to identify valid tuples of parameters. Unsafe link detection and action matching become more complex as well.

We adopted a hybrid approach that reduces the branching factor but which minimizes the complexity of the implementation and the performance overhead. In our approach, when adding a new action we instantiate only the parameters that are needed in order to satisfy the open condition. Note that adding an action may result in new open conditions corresponding to the unsatisfied preconditions of this new action with uninstantiated variables. Unlike the lazy approach described above, we do not allow these uninstantiated variables to be propagated to new actions at a subsequent step. Instead, when we select an open condition that is uninstantiated, we generate instantiations at that time. Our approach reduces the branching factor because the number of variables in an open condition is low for provisioning (typically one or two). Further, the need for variable binding constraints is minimized because uninstantiated variables are not propagated. This significantly reduces the complexity of the implementation and the performance overhead.

Model Guided Variable Instantiation We take advantage of knowledge on resource and data center configurability constraints to minimize the number of tuples created when binding variables in an open condition. For example, the relationships between servers and their NICs, and also typically the relationships between the NICs and the switch ports to which they are connected, are all fixed. These fixed relationships limit the number of resources that need to be considered when instantiating an action.

As an example, consider the provisioning operation `addNetworkInterface` shown on the left side of Fig. 4, and its precondition `(established ?scn ...)`, where `scn` is a relationship of type `SystemContainsNic`. Without our strategy, the planning algorithm would generate all pairs of systems and NICs in the data center, 8 combinations will be generated for the data center model piece

shown in Fig. 4 (types and identities of relationships are omitted for simplicity). Most would lead to a search dead end since no action exists that can change the fixed relationships between servers and their NICs. Not only are a large number of possible instantiations generated, but they are not immediately eliminated from consideration. Using our strategy, on the other hand, only pairs connected by a relationship will be considered. Specifically, only the following pairs will be generated by the planning algorithm when instantiating this action: `(EJB-server, nic4)`, `(EJB-server, nic5)`, `(Data-server ,nic1)`, and `(Data-server, nic2)`. Clearly, if one of the variables in this example were already instantiated, the number of tuples is further reduced. As a further enhancement of this strategy, we not only look at the preconditions of the current action when instantiating variables, but also at other preconditions, associated with relationships that are know to be fixed, where the variables appear.

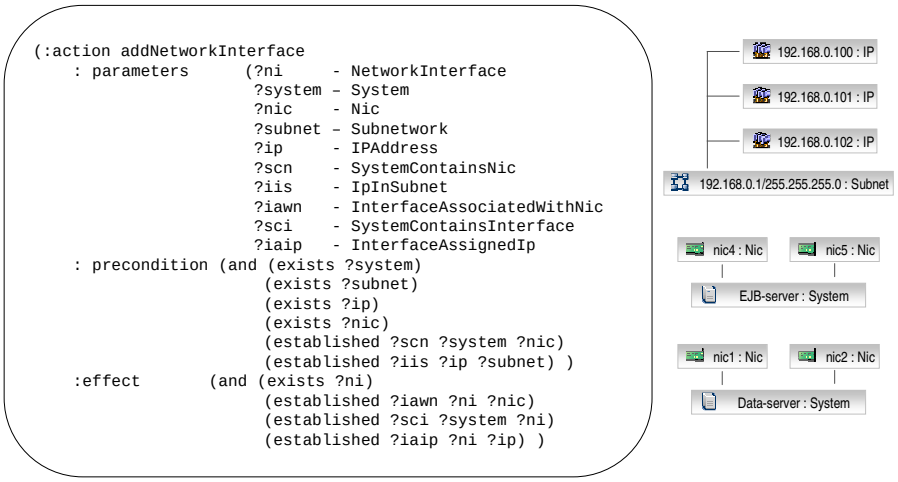

Fig. 4. Specification of the `addNetweorkInterface` provisioning operation and part of initial state showing two servers, their NICs and several IP addresses.

Deferred Variable Instantiation A deployment topology describes how resources in a data center should be configured. While the deployment topology identifies all the resources needed, it may not have fully selected them. Recall that a data center may implement policies that prevent some types of resources from selection until deployment time. For example, to configure a network interface, it is necessary to have an IP address. While the planner knows, based on the data center model, what IP addresses are in use, it does not have control over the selection of IP addresses, as data center policies typically determine the selection at deployment time using a provisioning operation. This restriction prevents the planning system from instantiating the IP address in other net-

work configuration actions, such as to configure network interfaces, routing, and access control. To address this challenge, we introduce the concept of *deferred variable instantiation*. For variables that can only be instantiated at deployment time, we create *placeholders* that represents the instantiated variables. Such a substitution can, however, take place only when the following conditions hold:

1. There is a provisioning operation (action) that can create a new unique instance of the required variable. For example, to resolve an IP address, the provisioning system may have an operation `getIPAddress` that generates a valid IP address.
2. The variable does not change in value once it is created.

The placeholder represents the output of a particular provisioning operation. It is treated as a read only instantiated variable that can be used as a parameter to other operations.

5 Prototype Implementation

We implemented a prototype of the workflow generator architecture and planning algorithm described in the previous sections. For the role of the workflow generator, we developed a custom Java-based partial order planner. The planner was implemented with configurable support for our domain specific variable instantiations, flaw selection, and deferred variable instantiation. For the role of the workflow engine, we used the IBM Tivoli Provisioning Manager (TPM) [2] v.3.1 product. TPM workflows are parameterized with strongly typed objects defined in a data center model (DCM). The DCM schema is DMTF CIM-based [14]. DCM instance data is populated manually, or automatically by discovery tools. TPM provides a device abstraction layer whereby logical device operations (LDOs) are declared against DCM types. This layer enables users to define workflows that can operate over different vendor implementations of a logical device. DCM objects are bound to device drivers that bundle device-specific implementations of logical device operations. We modeled a subset of the TPM LDOs relating to network configuration, as planner actions. We also implemented an importer from the DCM object model to logical representation. Finally, we implemented an Eclipse-based graphical user interface, with views for planner operations, initial and goal states, and generated operation partial orders.

Our prototype was also integrated with the SPiCE (Service Plan Composition Engine) model-driven data center design tool [3]. Using SPiCE, users could customize a logical application structure with deploy-time choices, and automatically generate the desired state of the data center. Our worflow generator would then be invoked, in the same Eclipse shell-sharing environment, to generate the the partial order of TPM LDOs required to provision the data center changes. A TPM workflow exporter was implemented to convert the planner's output to the TPM workflow language. Generated TPM workflows were submitted for execution to the TPM deployment engine.

6 Empirical Evaluation

We evaluated our prototype using the desired network structure of a three-tier clustered application consisting of a web, business logic, and data tier. An example of this structure is depicted in Fig. 5. An external browser system was defined to model remote access to the web tier. Browser traffic would be routed through a firewall, connected to a load-balancer, spreading requests across the servers in the web cluster. Web tier servers would invoke business logic functions by routing traffic over an internal firewall. Load balancing on the business tier would be performed at the application-level. Business tier servers were modeled as dual-homed, connecting to the data tier through another firewall. The figure is a screen shot of the SPiCE visualizer and depicts a filtered view over the desired state topology. Server, router and load-balancer network interface card (NIC), IP interface configuration, and routes were hidden. The switch and switch ports over which the VLANs were defined were also hidden.

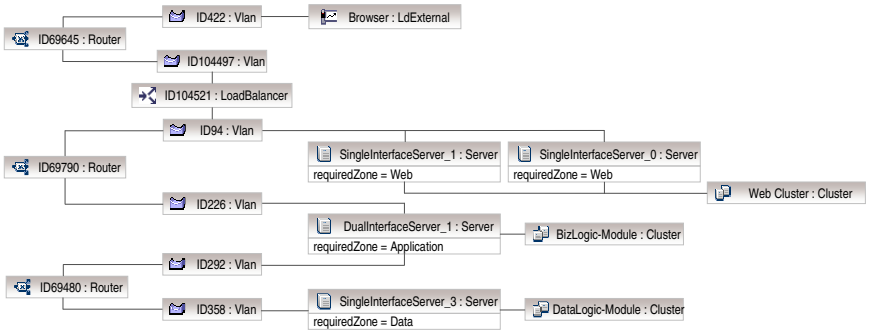

Fig. 5. Sample three-tier application network structure.

The desired state models were varied in the number of servers per cluster. The models were generated by defining a SPiCE logical application structure and varying the cluster size. For our infrastructure modeling, we created a parameterized DCM generator that created data centers with the requisite number of switches, routers, load-balancers and servers. The number of NICs on each device was also parameterized. The generated data center resources were instrumented by a simulator device driver provided by TPM.

We modeled the network device logical device operations (LDOs) for creating a VLAN on a switch, assigning a port to a VLAN, creating IP network interfaces on systems (routers and servers), creating routes, access control lists, and virtual IP addresses, creating clusters and free-pools, and adding servers to clusters and free pools. We also modeled resource-selection operations for selecting IP addresses/subnets, assigning tiers, and determining cluster expansion sizes (for support of TPM dynamic orchestration features). Figure 6 lists a partial workflow generated for a minimal topology. The workflow represents a serial ex-

ecution of the partial order generated by the planner (topological sort). It starts by creating the spare pool, customer and subnet logical resources defined in the desired state. Next it obtains a unique VLAN ID, and creates a new VLAN with the specified ID in the switch identified in the desired state. It configures the switch port to the newly created VLAN ID (must precede VLAN creation on switch). Note that this operation required the switch module containing the port as a parameter. This information was missing from the desired state. The planner used the module port lookup LDO to obtain the required parameter.

```
workflow SPiCEDeployOneTimeWorkflow LocaleInsensitive

 // variable declarations deleted ...

 CreateSparePool( "Pres-Module_pool", ID211)
 CreateCustomer( placeholder_2, ID213)
 CreateSubnet( ID92 )
 GetUniqueVLANNumber( placeholder_7)
 CreateVLAN( "1209", "ID68", placeholder_7, ID68)
 GetIPAddress( placeholder_10, "1219", ID92)
 AddNetworkInterface( "1218", "1219", ID92, placeholder_10, DT_ID75)
 CreateApplication( "MyAppStagEnv", ID213, ID214)
 GetPortModule( placeholder_24 )
 MovePortToVLAN( "1209", "1202", Vlan_DT_ID68, placeholder_24, "1")
 CreateRoute( "1218", ID152, ID94, ID75, ID193)
 GetClusterTier( placeholder_20 )
 GetClusterMinServers( placeholder_21 )
 GetClusterMaxServers( placeholder_22 )
 CreateCluster( "Pres-Module", placeholder_20, placeholder_22, placeholder_21, ID214, ID148 )
 AddServerToCluster( ID148, "1227" )
 AssociateClusterToPool( ID148, ID211 )
```

Fig. 6. A example of a generated workflow.

We focus on scalability studies that show our domain specific enhancements, described in Sect. 4.2, scale well and are better performing than the generic POP algorithm. We present two scalability experiments using the workflow generator. They investigated the scalability of the generator in terms of the infrastructure size and the size of the three-tier application desired state.

First, we varied the number of available resources in the data center keeping the number of resources in the desired state constant. Because the desired topology was unchanged, the number of provisioning operations was constant, modulo the selection of different servers. Therefore, the experiment measured the performance of the planner's variable instantiation. Figure 7 shows the planning time for infrastructures containing between 10 and 250 servers. We benchmarked our POP planner performance with domain specific optimizations enabled and disabled. The results show that the effect of our variable substitution optimizations result in a significant speedup for the base case, and are significantly less sensitive to infrastructure size increases.

In our second scalability experiment, we varied the number of resources in the desired state by increasing the number of servers in each tier cluster. We varied the total number of servers from 4 to 128. In this case, we kept the number of

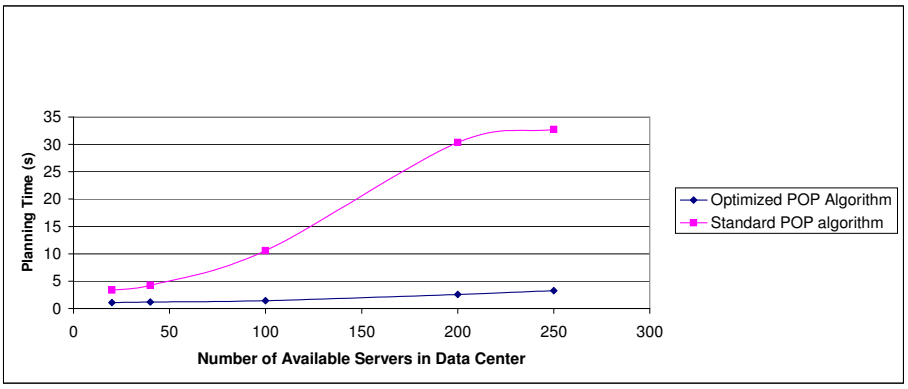

Fig. 7. Planning time vs. the number of servers in the infrastructure.

resources in the data center constant at 250 servers. Under these conditions, the number of provisioning operations will grow, resulting in a larger search space. Figure 8 shows the results of benchmarking our POP planner with optimizations enabled and one point with optimizations disabled. For problems with more than 4 servers we were unable to obtain solutions using the unoptimized planner.

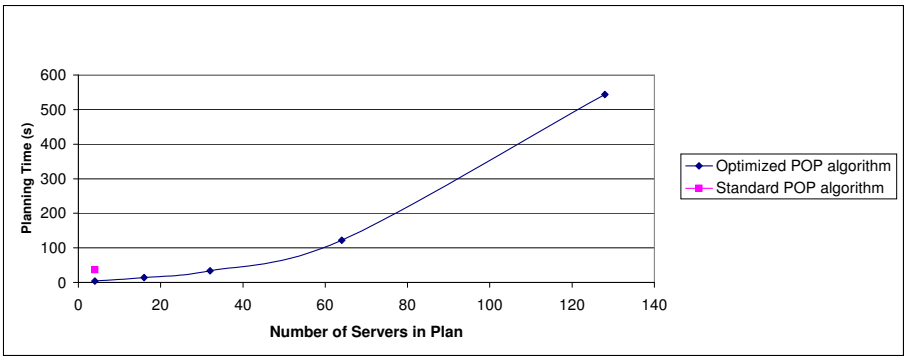

Fig. 8. the number of generated partial plans vs. the number of servers in the deployment topology

7 Related Work

Planning techniques are increasingly being adopted for distributed system management. Several projects have recently used planning techniques for the deployment of component-based applications [15–17], the composition of web services [18], and the management and execution of scientific workflows in the

Grid [19, 20]. There are several main differences in focus between these works and the work presented in this paper. All of these works focus on software (or service) level configurations. In contrast, our work is focused primarily on the low level network configuration aspect that is driven by the application requirements. The main usage of planning in these works is the optimization of resource placement, resource usage, and/or execution time, where a simplified model of the provisioning and configuration actions is assumed. In contrast, in our work we assume that an input desired state identifies the selected resources. Rather, we focus on the correct ordering and instantiation of complex real world provisioning and configuration actions with multiple preconditions and effects on the system state, and with a large number of input parameters.

Specifically, in [17], the authors addresses the issue of resource-aware deployment of component-based distributed applications in wide-area systems though planning. They provide a model, called the component placement problem (CPP), that describes the placement of application components onto computational, data, and network resources across a wide-area environment subject to constraints. The planner generates a plan of application components placement on a set of networked nodes. The work does not address the provisioning operations necessary to implement the solution and their ordering.

The CHAMPS system [16] focuses on Change Management, a process by which IT systems are changed through software upgrades, hot fixes, or, hardware changes. Upon the reception of a request for change, CHAMPS assesses the impact of the change and generates a change plan (as a BPEL workflow). Planning is used to optimize resource selection and execution time, while it is assumed that needed provisioning operations and their temporal dependencies are known.

Several techniques have been suggested to limit the search space when using planning for the dynamic composition of web services. Similar techniques might be applicable to our domain such as using business rules to guide the search space [21] and adopting a mixed-initiative approach where users can interact with the planner to drive the workflow composition process [22].

It is widely agreed on that proper modeling of the planning domain is key for correct and efficient planning. Several efforts have manually encoded the necessary domain knowledge [23, 24]. This is error prone and requires extensive efforts which hinder the practicality and adoption of the approach. In this work, we advocate the usage of object models for representing the current and goal state. In addition, we sucessfully integrated our planner with a modeling tool that generates an object model representating the desired state [3], and with a provisioning engine that provides the current state.

8 Summary and Future Work

Separation of deployment concerns is key to improving data center reliability, as well as reducing capital and operational costs. Emerging model driven technologies are showing great promise in the direction of weaving functional application

aspects, with non-fuctional aspects such as security, performance and availability, and data center resource availability and policies. Bridging the model to provisioning system gap is a key challenge in releasing the value of these tools. In this paper we demonstrated that with proper optimizations, planning algorithms can provide this bridge. Our initial results focused on generating network provisioning workflows driven by application requirements. Future work will focus on extending the operational models to the software domain. Resource selection can be performed in various stages of the deployment design process, and future work will examine usability and performance implications of alternatives. Existing workflows can be mined for dependencies, and compared to generated workflows to detect unusual ordering patterns. Desired state models may introduce non-functional operational dependencies, which should be honored by the planner.

9 Acknowledgements

The authors would like to thank the following people for discussions and ideas over the past three years: Giovanni Pacifici, Lily Mummert, John Pershing, Hendrik Wagner, Aditya Agrawal, Guerney Hunt and Andrew Trossman.

References

1. Brown, A.B., Keller, A., Hellerstein, J.: A model of configuration complexity and its applications to a change management system. In: International Symposium on Integrated Network Management. (2005)
2. IBM: Tivoli provisioning manager (http://www-306.ibm.com/software/tivoli/products/prov-mgr/) (2006)
3. Eilam, T., Kalantar, M., Konstantinou, A., Pacifici, G.: Reducing the complexity of application deployment in large data centers. In: International Symposium on Integrated Network Management. (2005)
4. Eilam, T., Kalantar, M., Konstantinou, A., Pacifici, G., Pershing, J., Agrawal, A.: Managing the configuration complexity of distributed applications in internet data centers. IEEE Communication Magazine **44**(3) (2006) 166–177
5. Felfernig, A., Friedrich, G.E., et al.: UML as a domain specific knowledge for the construction of knowledge based configuration systems. In: SEKE'99 11th Int. Conf. on Software Engineering and Knowledge Engineering. (1999)
6. Taylor, R., Frank, R.: Codasyl data-base management systems. ACM Comput. Surv. **8**(1) (1976) 67–103
7. Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL—the planning domain definition language (1998)
8. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artif. Intell. **90**(1-2) (1997) 281–300
9. Weld, D.S.: An introduction to least commitment planning. AI Magazine **15**(4) (1994) 27–61
10. Minton, S., Bresina, J.L., Drummond, M.: Total-order and partial-order planning: A comparative analysis. Journal of Artificial Intelligence Research **2** (1994) 227–262

11. Knoblock, C.A., Yang, Q.: Relating the performance of partial-order planning algorithms to domain features. SIGART Bulletin **6**(1) (1995) 8–15
12. McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91). Volume 2., Anaheim, California, USA, AAAI Press/MIT Press (1991) 634–639
13. Penberthy, J.S., Weld, D.S.: UCPOP: A sound, complete, partial order planner for ADL. In Nebel, B., Rich, C., Swartout, W., eds.: KR'92. Principles of Knowledge Representation and Reasoning: Proc.s of the 3rd Int. Conf. Morgan Kaufmann, San Mateo, California (1992) 103–114
14. Force, D.M.T.: Common Information Model (CIM) Standards (http://www.dmtf.org/standards/cim) (2006)
15. Arshad, N., Heimbigner, D., Wolf, A.L.: Deployment and dynamic reconfiguration planning for distributed software systems. In: 15th IEEE Int. Conf. on Tools with Artificial Intelligence, IEEE Press (2003) 39–46
16. Keller, A., Hellerstein, J., Wolf, J., Wu, K., Krishnan, V.: The champs system: Change management with planning and scheduling. In: IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), IEEE Press (2004)
17. Kichkaylo, T., Karamcheti, V.: Optimal resource-aware deployment planning for component-based distributed applications. In: HPDC '04: 13th IEEE Int. Symp. on High Performance Distributed Computing (HPDC'04), Washington, DC, USA, IEEE Computer Society (2004) 150–159
18. Su, X., Rao, J.: A survey of automated web service composition methods. In: SWSWPC 2004: First International Workshop on Semantic Web Services and Web Process Composition. (2004)
19. Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Agarwal, A., Mehta, G., Vahi, K.: The role of planning in grid computing. In: 13th International Conference on Automated Planning and Scheduling (ICAPS), Trento, Italy (2003)
20. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., Koranda, S.: Mapping abstract complex workflows onto grid environments. J. Grid Comput. **1**(1) (2003) 25–39
21. Yang, J., Papazoglou, M.P., Orriëns, B., van den Heuvel, W.J.: A rule based approach to the service composition life-cycle. In: WISE, IEEE Computer Society (2003) 295–298
22. Kim, J., Spraragen, M., Gil, Y.: An intelligent assistant for interactive workflow composition. In: IUI '04: 9th international conference on Intelligent user interface, New York, NY, USA, ACM Press (2004) 125–131
23. Wu, J., Sirin, E., Hendler, J., Nau, D., Parsia, B.: Automatic web services composition using shop2. In: 2nd Int. Semantic Web Conference (ISWC). (2003)
24. McIlraith, S., Son, T., Zeng, H.: Semantic web services (2001)