RC25138 (W1104-038) April 11, 2011
Computer Science

IBM Research Report

Migration to Multi-Image Cloud Templates

Birgit Pfitzmann, Nikolai Joukov
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Migration to Multi-Image Cloud Templates

Birgit Pfitzmann, Nikolai Joukov
IBM T. J. Watson Research Center, Hawthorne, NY USA
{bpfitzm,njoukov}@us.ibm.com

***Abstract*—IT management costs increasingly dominate the overall IT costs. The main hope for reducing them is to standardize software and processes, as this leads to economies of scale in the management services. A key vehicle by which enterprises hope to achieve this is cloud computing, and they start to show interest in clouds outside the initial sweet spot of development and test. As business applications (e.g., a travel application or a web catalog) typically contain multiple images with dependencies, e.g., in a 3-tier architecture, one is starting to standardize on multi-image structures. Benefits are ease of deployment of the entire structure and consistent later management services for the business applications.**

Enterprises have huge investments in their existing business applications, e.g., their web design, special code, database schemas, and data. The promises of clouds can only be realized if a significant fraction of these existing applications can be migrated into the clouds. We therefore present analysis techniques for mapping existing IT environments to multi-image cloud templates. We propose multiple matching criteria, leading to tradeoffs between the number of matches and the migration overhead, and present efficient algorithms for these special graph matching problems. We also present results from analyzing an existing enterprise environment with about 1600 servers.

***Index Terms*—IT services, management costs, migration, clouds, multi-image templates;**

I. Introduction

IT management costs are the dominant cost of IT and still on the rise. Hence a key issue for IT services organizations, both in-house and as special IT services providers, is to reduce these costs. The main approaches are standardization and subsequent automation. For both, cloud computing is the current main hope. Key differentiators are standardization of images including their software stacks, and rapid deployment. The former promises to significantly decrease IT management costs, the latter reduced power, server costs, and data center footprints beyond the gains due to static virtualization. While the performance questions have already gained significant interest in the literature, the difficulties and benefits of standardization have been investigated much less thoroughly, although ultimately, given the dominance of services costs, one is hoping for greater savings from them. We will therefore focus on the standardization aspect.

Typical current business applications are not monolithic, but built in distributed ways, in particular often as 3-tier structures consisting of web servers, application servers, and databases. By a business application we mean a set of interacting software components that perform some task together, and are typically managed and in particular tested together. An example of a business application is a travel reimbursement application consisting of a web interface for entering travel expenses, an application server that validates limits and handles approval workflows, and a database that contains the persistent copies of the expense reports in all states. Dependencies between software components in a business application, as well as with other components that may reside on the same servers or even in the same application servers or web servers, are a key complexity factor of current IT management, in particular as they are not always well documented. Hence one of the ideas to minimize service costs via clouds is to provide standardized multi-image templates for typical business application structures, in particular 3-tier structures, and to manage these in a consistent way with metadata, image libraries, and deploy scripts. Clouds with multi-image templates can be seen as a specific form of PaaS (platform as a service).

To gain wide adoption for such clouds, we must devise methods to migrate a significant subset of existing business applications to these multi-image templates. Enterprises have huge amounts of existing software in these business applications, e.g., scripts on the web servers, Java code in the application servers, and transactions and reporting queries on the databases, as well as actual data. Only a very small percentage of business applications is written new per year, compared with what is retained unchanged or carefully upgraded. For instance, many applications get regular hardware refreshes and software upgrades. Virtualization has only gained its current real traction in the business world because physical-to-virtual (P2V) transformation has become a relatively cheap and low-risk operation. In comparison, e.g., SOA (service-oriented architecture) has only reached rather small coverage because it usually requires a significant amount of reprogramming of the business-application-specific code.

We therefore want to explore the migration of existing business applications to clouds with multi-image templates. We are not aware of any prior literature in this space, and we expect that significant additional work will be needed over the next years to fully optimize the tradeoff between migration costs and the level of attained standardization. In other words, the goal is to migrate as many existing business applications as possible to more standardized structures, while keeping the costs for the necessary changes small enough so that they amortize within reasonable time via cost savings in IT services.

We concentrate on three-tier systems as those promise the largest coverage for initial multi-image templates.

In Section II, we introduce our problem setting in more detail and give examples. We describe our general method in Section III, a core algorithm in Section IV, and more variants in Section V. We evaluate a real enterprise IT environment in

Section VI, and discuss related literature in Section VII. We conclude in Section VIII.

II. Setting

We speak of a source system, a cloud offering, and a target system. The source system is the existing enterprise IT environment. The cloud offering includes a catalog of multi-image templates from which cloud users can choose. A multi-image template is a structure of metadata about several images and their relations. A cloud offering also includes actual virtual-machine images according to the catalog templates. These images are stored as sets of files in an image library and can quickly be copied to sets of real servers. I.e., when a user selects a multi-image template, several images get deployed, and they are ready to communicate. Under the covers, deployment scripts may be used to update the images after copying, e.g., with concrete addresses for their communication. The target system is the result of our intended transformation, i.e., it consists of instantiations of the multi-image templates, and after the migration (in contrast to a green-field deployment), it will also contain the business-application-specific code and data of the subset of the source system that was chosen for migration. For instance, actual Java modules will have been migrated into an application server that was already on one of the images in the image library, and that was already configured to communicate with a database instance on another image from the same multi-image set, into which actual data have now been migrated.

For the matching algorithms, it is useful to distinguish software installs, services, and objects, with a natural inclusion relationship. An install is a software installation as produced by an installation process. A service is an instance set up to serve requests, e.g., a web server, an application server, or a database instance. Most software products, in particular in 3-tier architectures, allow multiple services per install. Running processes are typically associated with services, but in particular with databases, a service need not have constantly running processes. Objects mean everything that is handled by the services, e.g., web pages and scripts in a web server, applications and their modules in an application server, and databases, tables etc. in a database instance.

Dependencies between the components of a multi-tier application are usually set up at the service and object levels. In particular, dependencies of application servers are typically set at the service level, i.e., an application server contains configuration files that establish a connection to a database instance, message queuing service etc. It is also possible to establish those dependencies in proper configuration files at the object level, e.g., for Java EE applications and modules like war and ear files, but in real enterprises we see that very rarely. The fact that important dependencies are set at the service level rather than the object level is key to the vision of multi-image templates for clouds, because these templates and the corresponding ready-to-deploy images typically do not contain objects: Individual web pages, Java applications, and databases are established by the cloud users on these images, in our case via migration. This is because these are PaaS clouds, in contrast to SaaS (software as a service). Nevertheless, the base setup of the dependencies can be done at the service level and thus in the preconfigured images. An enterprise that is a cloud consumer may add specific images that also contain objects (content) to the catalog, e.g., for rapid scaling of a specific heavily used web server cluster, but this situation does not require matching. Hence we focus on matching existing source systems with multi-image PaaS templates that contain the install and service levels of software components.

It is preferable that each image in a multi-image template contains only one major software install or even only one major service (such as a web server, application server, or a database instance), because this significantly decreases management. For instance, all security settings on an image can then correspond to the one software without policy merging, and there is no resource contention or other negative influence among different software on an image. However, current physical servers commonly contain multiple software installations (to best utilize the hardware), and taking existing images apart is a significant migration effort and thus cost factor. Hence we also allow that images in the multi-image templates contain several major software installs or services.

As to the source systems, we assume that their current structure is known at the time of the matching algorithm. This will often require discovery prior to migration, as CMDBs (configuration management databases) are rarely deployed consistently throughout an enterprise.

A. Example Multi-Image Templates

Figure 1 shows examples of multi-image templates. Multi-image template M contains two images I and J. Image I includes a web server installation and an application server (APS) installation, with one service ("App server") set up. Image J includes a database (DB) installation, with one service set up. There are no applications or databases in the images I and J; these will be provided by the users of deployed instances of the multi-image template. Note that we speak of images in these examples although the matching algorithm actually works with the metadata about these images in the catalog. The graphical representation of the multi-image templates is only an example; the matching algorithms will typically work on database or XML formats. However, such graphics might be a good augmentation to current catalog formats.

Multi-image template M' is parameterized, i.e., there are parameters, i and j, that a user can select when choosing this template from the catalog. At the front end, the multi-image template M' includes an image H' with one web proxy. This web proxy serves as a load balancer for a number of identical web servers. Each of these web servers is an instance of image I', e.g., if a user needs ten web servers and thus chooses $i = 10$, then image I' is deployed ten times. These instances of web servers are the "same", which means that they are meant to get the same content. Similarly, at the backend there may be j databases, all replicas of each other. So, e.g., if a

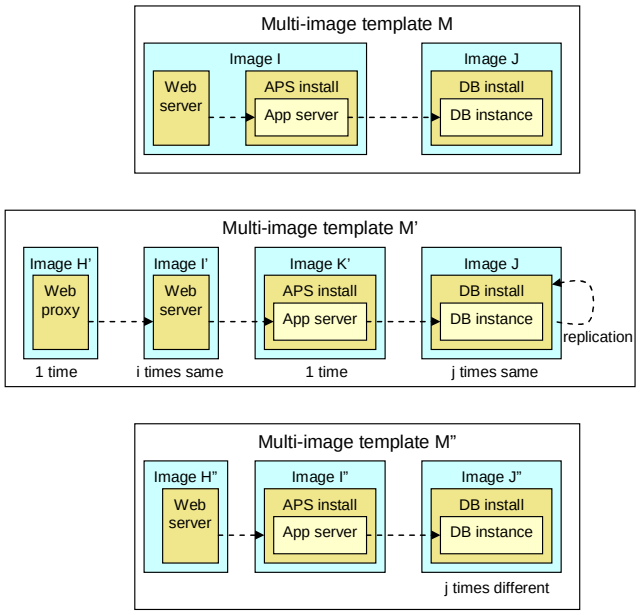

Fig. 1. Example multi-image templates.

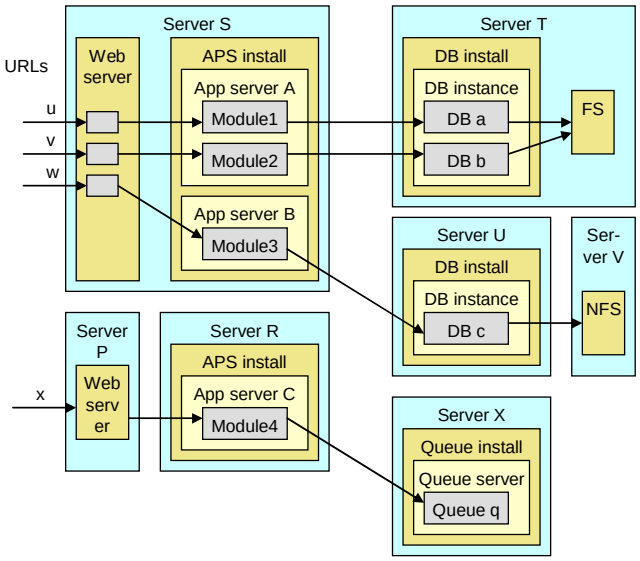

Fig. 2. Example source servers.

user chooses $j = 2$, then image J' is replicated twice. The dependencies are replicated as well, i.e., the web proxy will be linked to each web server, and the application server to each database. Additionally, the databases all replicate with each other.

Multi-image template M'' is also parameterized, but instead of replication, we now allow j *different* databases. Thus, a user choosing this multi-image template with $j = 3$ will get 3 instances of Image J'', and the application server will again be linked to each of them, but the databases will not be replica of each other, and this internally implies different settings in the application server. The user can put different content in each database instance.

B. Example of Source Applications

Figure 2 shows some source servers with software and dependencies. The example would typically be part of a large model resulting from the discovery phase of a migration process. There may be hundreds or thousands of physical or virtual servers. Again, the graphical representation is only used for illustration. Servers are indicated as S to X. E.g., Server S includes a web server and an application server (APS) install, and the APS install contains two services, which are application servers. As this illustrates a working source system, there are URLs implemented on the web servers and modules in the application servers. We have simplified the figure a little compared with actual discovery results, e.g., one would see applications between the application servers and the modules, and a URL on server P.

C. Initial Matching Thoughts

Server S is somewhat suitable for matching with image I, i.e., within a multi-image matching method as described below one may consider whether one can migrate server S to an instantiation of image I, assuming servers related to server S can be migrated to the remaining images in multi-image template M. An exact match between a source application and a multi-image template is preferable. However, many source applications will not have an exact match. For example, server S differs from image I by having two app servers A and B in its APS install. Furthermore, server S is linked to database instances on two servers T and U, while multi-image template M only includes one such image. These situations will be addressed below. Without app server B and its dependencies, the server pair (S, T) would perfectly match multi-image template M. Here, we assume that file systems (FS) are not explicitly represented in multi-image templates, i.e., the component FS on server T does not matter. Such strategies of what does and does not matter are also discussed below.

Alternatively, servers S, T, and U would match multi-image template M'' if either app server B and the dependency on server V were not there, or one is allowed to add them to M'', and if one is allowed to split the web server and the application server install from S onto two images (I'' and K'').

III. Overall Method

We now give an overview of our matching method.

A. Flow Diagram

Figure 3 shows a flow diagram for our migration planning using multi-image templates.

In Phase 1, source software components, their dependencies, and servers that these software components are on are discovered. As we want to match with details of multi-image templates, we need significantly deeper knowledge about the source systems than in a pure physical-to-virtual transformation. Discovery typically includes both network traffic analysis and studying the configurations of operating systems and software. There are several commercial products in this space, and also ongoing research, e.g., [4, 11, 12, 21]. A discovery

tool may already be in place in the source systems, or may be deployed for the migration. While automatic discovery is more precise, the following steps also work if discovery is manual, e.g., by asking application owners.

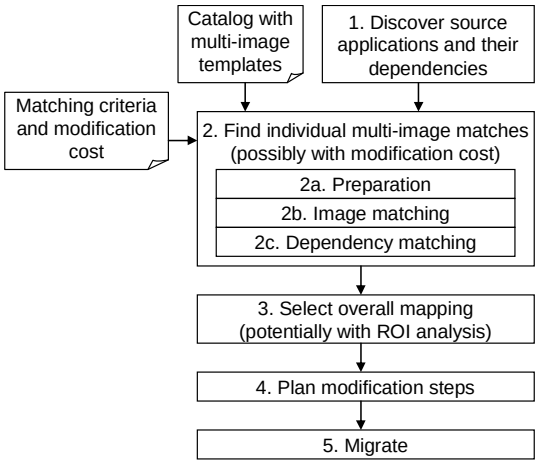

Fig. 3. Method overview.

In Phase 2, potential individual multi-image template matches are found. Individual means that in this phase, we look for all potential matches, not bothering whether some of them overlap in the source system. A match means that a sub-structure of the source system is found that is similar to a multi-image template in the catalog. This phase is governed by matching criteria that determine what "similar" means. Clearly, an exact match is optimal and corresponds to a substructure in the source system that gives topologically the same picture as one of the multi-image templates. We precisely define this below, and present a set of weaker matching criteria. The matching criteria may come with costs, because everything that is not an exact match requires modifications later, or wastes a feature provided by the multi-image template.

In Phase 3, an overall mapping is selected from the potential matches that resulted from Phase 2. This selection is needed because the potential matches may be overlapping or alternatives to each other. We aim at choosing an overall mapping with broad coverage and few necessary modifications to the source systems or templates. If we have quantitative estimates of modification costs, we can treat this as an optimization problem. Source images that were in no matching sub-structure, or that end up without matches in the overall mapping, may be mapped to individual images that also exist in the catalog, or have to be migrated purely as physical-to-virtual. E.g., server X in Figure 2 has no match in any multi-image template from Figure 1.

Actual modification steps, corresponding to the differences between source sub-structures and multi-image templates that the matching criteria may have allowed, are planned in Phase 4. In Phase 5, the actual source system is migrated into a cloud using the selected multi-image templates: The multi-image sets corresponding to the chosen templates are retrieved from the image library, instantiated on real servers using their standard deployment scripts, and potential modifications to their configurations are made as planned in Phase 4. Then, application-specific code and data are migrated from the source systems onto these new target instances, with potential source modifications planned in Phase 4.

Note that while we primarily target clouds with real multi-image libraries, our matching algorithms can also be used in scenarios without actual prebuilt images corresponding to all the multi-image templates, and where the templates guide an on-the-fly model-based deployment of the software on more basic catalog images.

B. Example

We now show how this method applies to the source systems from Figure 2 and the example multi-image templates in Figure 1. The multi-image templates M, M', M'' might in reality be three of many in the catalog. The source applications shown in Figure 2 would be part of the result of discovery. In the matching step, we find individual sets of servers in the source applications that match individual multi-image templates such as M, M', or M''. In the selection step, we may select from several possible matches. The matches may be scored to determine a best match for the source system.

If we only match self-contained server groups that have no dependencies other than those present in a multi-image template, this selection of an overall mapping can be done per connectivity component of the source environment. If we allow additions of dependencies to target images, there may be overlapping potential matches after the matching step and we may use global cost computations. E.g., out of the three servers P, R and X in Figure 2, a multi-image template M_{front} may match P and R, while another multi-image template M_{back} matches R and X. Then, a decision needs to be made whether it would be easier to use M_{front} and an additional image with a queue server, or to use M_{back} and an additional image with a web server. Matching criteria govern how much one is willing to change between the source system and the multi-image templates.

C. Graph Models

For the algorithmic matching, we assume that the source systems and multi-image templates are represented as labeled graphs. Nodes in the source system have at least two labels, a type such as "APS install" and a node name such as "Module 1" in the example. Nodes in the multi-image templates have at least a type label. We require that the same type system is used for multi-image templates and source systems. In practice, we may first need a terminology mapping to get the labels from a discovery tool or CMDB and those of a cloud metadata schema to agree.

For edges, it is useful to distinguish inclusions from other dependencies. This is shown as nested boxes versus arrows in Figures 1 and 2. Inclusions are used for components that run "inside each other", offering each other an environment or abstraction layer, e.g., a database in a database instance based on a database installation. Inclusions may imply co-location.

Dependencies (arrows) may occur and thus be represented at different levels of inclusion. In multi-image templates, dependencies will mainly occur for services. In source systems, they may also occur for inner objects (such as individual modules and databases) or only be known at the server level (e.g., if observed via network statistics). We assume in the following that all server-level dependencies that can be associated with known software on that server have in fact been associated as precisely as possible.

Hence formally we have a structure (N, I, D, L) where N denotes nodes, $I \subseteq N \times N$ inclusions, $D \subseteq N \times N$ (other) dependencies, and $L : N \rightarrow T \times Nm \times Conf$ denotes the node labeling, where T is the set of node types, Nm the set of node names, and $Conf$ the set of configurations, with $\epsilon \in Nm$ and $\epsilon \in Conf$ denoting that no name or configuration is given.

Matching between source systems and multi-image templates is a kind of subgraph isomorphism problem between these graphs. How similar a source subgraph and a multi-image template graph have to be (i.e., really isomorphic or not quite) in order to be declared a match is determined by the matching criteria from Figure 3. Three kinds of matching criteria govern how closely the node labels have to match, how similar the inclusion graphs per image have to be, and how similar the dependency graphs have to be. The tradeoffs to be made when selecting criteria for a specific situation include

- ease of migration; this is best with close matches that do not require source changes,
- steady-state simplicity; this is best with close matches that do not require target changes, and
- coverage, i.e., more existing workloads can be migrated; this is best with looser matching.

A tool should offer a range of matching criteria, and one may apply them to the same source system and multi-image templates to find the lowest cost in Phase 3.

IV. Detailed Algorithm for Precise Matching

We now describe the individual matching from Phase 2 of the overall method in greater detail as a preparation phase, an image matching phase, and a dependency matching phase.

In this section, we do this for precise matching. By this we mean that the multi-image templates are not parameterized, and that the source structure and a matched multi-image template must have precisely the same software components (as far as they are considered on the templates, typically installs and services) with the same inclusions and dependencies. We only allow flexibility in the node matching, and in the treatment of dependencies on infrastructure software.

The matching problem belongs to the class of graph isomorphism problems. Graph isomorphism is in NP, and not known to be either in P or NP-hard. However, in most practical cases, we can solve the multi-image matching problem quite fast, because the multi-image templates are not large. Our examples with three to five main software installation types (unparametrized in this section, parametrized in the next section) are realistic. Furthermore, the nodes have types. This reduces the possibilities for matching very significantly.

A. Matching Criteria

The multi-image matching builds upon matching individual software components, i.e., graph nodes. This is governed by matching criteria of the first kind. At the end of this subsection, we describe the matching criteria for dependencies on infrastructure, which belong to the third kind.

Type Labels. Clearly, we have to match the type labels of the source nodes and the template nodes, e.g., we cannot match a database install to a web server install. It is best to have a hierarchical type system so that we can choose matching criteria with varying strictness, e.g., as follows, with consequences on subsequent the migration complexity:

1) Only software components of the same product and exact version match, e.g., a product's version 9.9 with version 9.9.
2) Software components of the same product and major version match, if the minor version of the source component is smaller or equal than that of the template component. E.g., source versions 9 to 9.8 also match template version 9.9, but not vice versa. This requires a software upgrade for any version other than 9.9, but in many cases this is an easy operation provided by the vendor.
3) All source versions match any newer version of the same product. This requires more complex upgrades for older source versions.
4) Even different products of the same class, e.g., databases from different vendors, match.

Each matching criterion defines a relation "$\leq$" over the type set T where $\tau_s \leq \tau_t$ must hold for a source type τ_s and a target type τ_t for a match.

We treat operating systems like installs, possibly with their own matching criterion because operating system upgrades or changes have particularly large consequences.

Name Labels. We do not match the name label of the source systems, as user-given names seem too unlikely to ever match names in templates. Mostly, names are given to objects, e.g., exact names of databases, and object names will come over in the migration of application-specific code and data. Installs are very rarely named, but installation directories may change and this induces modifications for Phase 4. Services are sometimes named, and if those change to names of corresponding services on templates, one needs to consider whether there are any uses of these names directly in objects that are migrated, and also plan those modifications in Phase 4. Formally, the non-matching may be done by using ϵ for all name labels in the template images, and defining that ϵ matches all names.

Node Configurations. Configuration files may also be compared if the configurations on the actual images in the image library are taken as somewhat prescriptive, in order to simplify later management, and thus at least partially prescribed in the template metadata. For example, if a source database instance defines a certain diagnostic level, a target image diagnostic level may be desired to be at least as good.

Configuration matching need not be considered in single-

image IaaS cloud migrations, since there is no existing middleware on the target images. However, it would also be useful in migrations with single-image platform-as-a-service cloud targets, and in our case when selecting single-image targets for those source servers that did not get a multi-image match. As matching criteria, one may define a "$\leq$" relation on configuration parameters, where $c_s \leq c_t$ must hold for a source parameter c_s and a target parameter c_t for a match.

Helper Software. Typical servers contain infrastructure programs like shells, Java runtimes, monitoring agents, and security software. One may or may not want to include these programs in the matching. For instance, if one desires to unify the monitoring infrastructure by the cloud migration, one will not match on the current monitoring agents and will not migrate them, but rather use those provided on the cloud images. However, one has to analyze whether there are modifications to plan in Phase 4. This may not be the case with OS-level monitoring software, but, e.g., with configurations of a host firewall software. Software that is not to be matched is deleted from the source graph now.

Dependencies on Infrastructure Servers. An IT infrastructure includes common services, e.g., DNS, LDAP, and print servers. Such servers cannot be put into each multi-image template. However, many source servers depend on these services. Hence, in the matching, dependencies on such services are not considered. This may be done by deleting all nodes of these types and all dependencies with them from the source graph. Alternatively, one may represent dangling dependencies in the multi-image templates, e.g., a dependency from an application server to a not-included LDAP server, meaning that this multi-image template can use a general LDAP server available in a target cloud. In the following, we use the first alternative.

B. Preparation Phase (2a)

In Figure 4, the preparation phase of a precise matching method is shown.

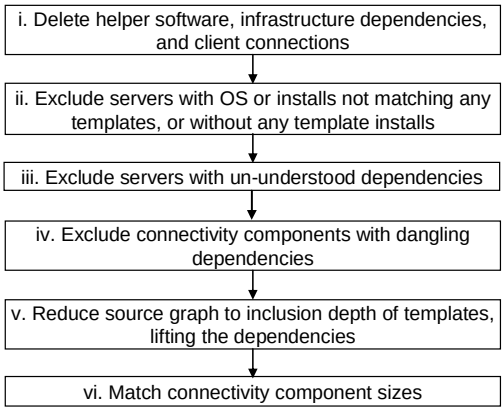

Fig. 4. Preparation phase for precise matching.

In Step i, delete not-to-be considered helper software, infrastructure servers, and dependencies on them from the source graph according to the matching criteria.

In Step ii, make a list or hash table SI of software installs occurring at least once in a multi-image template, and exclude all source servers that include an install $s \notin SI$. Further exclude source servers without any install $s \in SI$.

In Step iii, we exclude source servers that have a dependency on the server level that has not been associated with a software component on this server. This is typically a dependency found by network observation, and indicates that some software on this server was not discovered (neither as helper software nor as infrastructure software nor as a main software component) or sufficiently analyzed for dependencies to allow correct matching.

In Step iv, we divide the server level of the source graph into connectivity components. By server-level we mean that every dependency of two nodes is only considered as a server-to-server dependency in this division, so that we operate on a simpler server-only graph. A connectivity component is a set of nodes in a graph that have no link to nodes outside this set, but cannot be divided into smaller such sets. Figure 2, assuming nothing was deleted in Steps i to iii, has two connectivity components, servers S, T, U, V and servers P, R, and X. Connectivity components can be determined efficiently with well-known methods. Note that the source graph in this step may contain dangling dependencies, e.g., to other servers where no discovery was run, or to servers excluded in Steps ii and iii. Servers with dangling dependencies and others connected to them are also excluded in this step. Precise matching implies that every match has to be between a remaining source connectivity component and a multi-image template.

In Step v, we reduce the components on the source servers to the levels of depth included in multi-image templates, i.e., typically installs and services. If there are inner dependencies, they are first lifted to the next-outer remaining level, e.g., a dependency on a database becomes a dependency on the surrounding instance, and a dependency from a URL becomes a dependency from the surrounding web server. Multiple resulting dependencies between the same two components are reduced to one. (Or if dependencies were labeled, only dependencies with the same labels would be reduced to one.)

In Step vi, we exclude connectivity components whose size, i.e., the number of servers in them, is different from the size of each multi-image template.

Figure 5 shows the result of applying the preparation phase to the source system from Figure 2. As matching criteria for Step i, we declared that network file system (NFS) servers and file systems are infrastructure, and that external incoming dependencies on web server URLs are client connections and thus irrelevant (as they are not shown in the multi-image templates, but will obviously occur). Server X is excluded in Step ii because there are no Queue installs in the multi-image templates. Thus in Step iv also its connectivity component with servers P and R is excluded. The objects in the application servers and DB installs disappear in Step v because the multi-image templates only prescribe installs and services. The dependencies of these inner components, e.g., those between

Module 1 and DBa, and Module 2 and DBb, have been lifted to the containing services. The result is Figure 5.

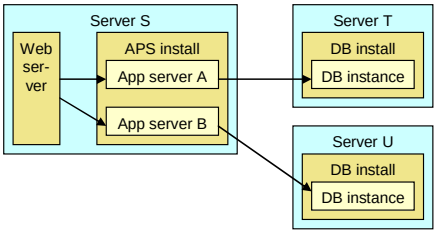

Fig. 5. Example source system after preparation phase.

Several of these preparation steps are not strictly needed. However, they are efficient, and in practice they significantly reduce the problem size for the following more complex steps. They also give simple human-understandable reasons for non-migratability of certain source systems. Furthermore, they can be used in a situation that may initially be most common, where a set of multi-image templates and single-image templates is chosen specifically for the standardization of a given source enterprise environment.

C. Image Matching Phase (2b)

In Phase 2b, we start matching each multi-image template M from the catalog with each source connectivity component SC that remains after the preparation phase. Let M consist of images $I_1, \ldots, I_n$, and let SC consist of servers $S_1, \ldots, S_n$. In this phase, we determine which pairs of images match; in Phase 2c we investigate the dependencies.

Hence for $i, k = 1, \ldots, n$ we want to determine whether I_i and S_k have the same included components. If yes, we determine the set Φ of possible mappings between the nodes of I_i and S_k. This is a tree isomorphism problem (because the inclusion relation of servers, install, services etc. yields a tree per image or server) with typically very few nodes. Methods to solve it include term unification with commutative operators. We present pseudo-code of such an algorithm, which we call $treematch$.

It is recursive, takes two tree nodes as input, and outputs "false" or ("true", Φ) where Φ is the set of possible mappings between the trees.

Initially, we call $treematch(I_i, S_k)$, i.e., with the images themselves as tree roots. A definition of $treematch(c, d)$ for arbitrary tree nodes c, d is given as follows:

- Let l, m be the number of children of c and d. If $l \neq m$, return "false". If $l = m = 0$, return ("true", $\{\epsilon\}$), where ϵ denotes an empty mapping. Else initialize Φ to the empty set.
- Explore each bijective mapping $\phi(c_1, \ldots, c_m) = (d_1, \ldots, d_m)$ from the children of image component c to the children of source component d where $type(c_i) \geq type(d_i)$ for all i:
  - For $i = 1, ..., m$ call $treematch(c_i, d_i)$.
  - If all m calls return ("true", Φ_i), augment the current set Φ by the following set of mappings:

$$\{\phi \cup \phi_1 \cup \ldots \cup \phi_m \mid \forall i = 1, \ldots, m : \phi_i \in \Phi_i\}.$$

- If Φ is still empty after the previous step, return "false", else return ("true", Φ).

Note that $treematch$, by itself, can be used for single-image matching as well as multi-image matching.

D. Dependency Matching (2c)

In the dependency matching phase, we take each possible node matching and determine whether the dependencies also match (i.e., are isomorphic). Let M and SC be as above.

We investigate every bijective mapping $\psi : \{I_1, \ldots, I_n\} \rightarrow \{S_1, \ldots, S_n\}$ where $treematch(I_i, \psi(I_i))$ returned ("true", Φ_i) for all i. In other words, ψ is a potential mapping of the servers after Phase 2b. Typically, there will be at most one such mapping for each pair of M and SC. The maximum of $n!$ mappings ψ can only occur if all the images in the multi-image template have the same components. Even this is not a problem for a small n like 3, 4, or 5.

Given ψ, we consider each combination $\phi_1 \in \Phi_1$, ..., $\phi_n \in \Phi_n$ of detailed image mappings, and let

$$\lambda = \psi \cup \phi_1 \cup \ldots \cup \phi_n$$

denote the overall mapping of all the nodes of all the images in M and SC. Typically, there will also be only one such mapping λ.

Next we determine whether the dependencies configured in the multi-image template are also present between the corresponding source components; we label such source dependencies as "used". For each overall mapping λ:

- Label all the dependencies in the source connectivity component SC as "unused".
- For each dependency (c, d) of the multi-image template M (where c, d are components of this multi-image template):
  - Check whether a dependency $(\lambda(c), \lambda(d))$ exists in the source connectivity component SC and is "unused". (If it exists, it is automatically "unused" as long as we have single dependencies between each pair of components.)
  - If no, mapping λ is not a match, and we abort it.
  - If yes, mark $(\lambda(c), \lambda(d))$ as "used".
- If the loop was not aborted, check whether all the dependencies of the source connectivity component are "used". If yes, the multi-image template M and the source connectivity component SC are a match with the overall mapping λ, else not.

E. Later Phases for Precise Matching

Phase 3, the selection of an overall mapping, is relatively simple in the case of precise matching. As entire connectivity components of the source system have to precisely match a multi-image template, the only possible ambiguity is if a source connectivity component matches two structurally identical multi-image templates with slightly different node details, according to the chosen matching criteria for nodes. E.g., if there are multi-image templates with two versions of a

database product, and we use loose matching of versions, there may be two matches for one source connectivity component SC. In such cases, we can compute modification costs for choosing a multi-image template M with node mapping λ from Phases 2b and 2c as

$$cost(SC, M) = \sum_{n \in M} mod\_cost(\lambda(n), n),$$

where n are nodes in the multi-image template M and $mod\_cost(x, y)$ denotes the costs for modifying node x into node y. We may simply compare the costs for the possible templates M for each SC individually. However, if we are still designing a good multi-image template catalog in parallel with analyzing the source system, we may add steady-state management costs for each multi-image template we need, say a constant cost. Then the costs to be optimized over all connectivity components SC_i, $i = 1, \ldots, \sigma$, with possible matches M_i become

$$\sum_{i=1}^{\sigma} cost(SC_i, M_i) + |\{M_i \mid i = 1, \ldots, \sigma\}|,$$

where $|\cdot|$ denotes the size of a set.

In the modification planning and migration (Phase 4 and 5), the configurations of source software components that were matched to a multi-image template are aligned with the configurations of the corresponding components of the multi-image template. The matching criteria used before should ensure that this is feasible. In particular, dependencies may be configured differently in the source systems than on the images in the templates, e.g., some addresses may be embedded in actual code, while the application servers in the image library would have their dependencies configured in XML files according to Java Enterprise Edition standards. To fully benefit from the multi-image templates, such cases should be found and remediated, e.g., as initiated for Java code in [12].

V. Other Matching Variants

In this section, we describe variants of the preceding algorithm.

A. Precise Matching with Parameterized Templates

As a second algorithm refining our overall method, we consider precise matching with parameterized multi-image templates, such as M' and M'' in Figure 1. The preparation phase is as for non-parametrized multi-image templates, except that the size-matching step vi is skipped. Then the following steps are performed for node matching and dependency matching. For each multi-image template M in the catalog with images $I_1, \ldots, I_n$, and each remaining source connectivity component SC with servers $S_1, \ldots, S_m$ with $m \geq n$:

- For $i = 1, \ldots, n$; $k = 1, \ldots, m$ determine whether I_i and S_k have the same included components, and the set Φ of possible mappings, with the same algorithm $treematch(I_i, S_k)$ as in the case with non-parameterized multi-image templates. (Recall that we currently only parametrize the number of entire images, not components on images; otherwise we would have to modify $treematch$ too.)
- For each partitioning $(S_{1,1}, \ldots, S_{1,m_1})$, $\ldots$, $(S_{n,1}, \ldots, S_{n,m_n})$ of $S_1, \ldots, S_m$ such that each $S_{i,k}$ matches I_i:
  - Construct the corresponding expanded multi-image template, i.e., with m_i copies of each image I_i, and all the corresponding dependencies.
  - Match the dependencies as in the non-parametrized case.

  This should yield zero or one matches if the parameterized multi-image template was defined so that no two separately listed images have the same component trees.

B. Less Strict Matching Variants

Several less strict matching criteria for Phases 2b and 2c are conceivable, with increased work needed in the actual migration.

Unused dependencies in the multi-image templates may be allowed. This means performing the dependency matching with λ^{-1} instead of λ and without a test whether all dependencies are labeled "used" at the end. This variant makes no real sense for multi-image templates with very few dependencies as in our examples M, M' and M'': If any of these dependencies is not used, one could better use two templates with fewer images each. In those cases our preparation phase would already exlude the potential mappings due to different sizes of the connectivity components. But this variant might make sense if there were multiple preconfigured dependencies between certain pairs of images. In the actual migration, the unused dependency configurations on the instantiations of the multi-image templates may be retained for potential future use, or deleted.

Unused components in the multi-image template may also be permitted. We may have a set of source servers that are a subset of the components of a multi-image template, e.g., a set like multi-image template M' but without the web proxy. Using multi-image template M' here may be considered a waste of components and may not be allowed (i.e., in most cases precise matching will be preferred regarding this aspect), but the method can be extended to accept the unused components. The method makes more sense than with the sample templates from Figure 1 if the templates include images with several major software components each, or if one starts also matching on minor software components such as compilers and infrastructure management software.

Additional dependencies on the source systems may be permitted. A typical multi-image template will not have dangling dependencies (except possibly to infrastructure servers as introduced above), i.e., all dependencies are only among the images of the multi-image template. One goal of the simplified management via multi-image templates is not to introduce additional dependencies that are not known in the templates. Therefore, we consider the dependency mapping described above, where every dependency of the source system must be

"used" at the end, preferable. However, this may be relaxed to permit a certain number of additional dependencies.

Server stacking or unstacking may be permitted. We may permit that software installations that are on different servers in the source system are stacked onto the same image in the target system, or vice versa. In Figure 2, we may permit that the web server from server P and the application server install from server R are moved onto the same image, thus making the union of these two servers similar to image I in multi-image template M. Or we may permit that the web server and the application server install from server S are split, thus making that part similar to images I' and K' of multi-image template M'. In this case, the algorithm $treematch$ can remain the same, but it is now called on installs, not on images. Similarly, the graph matching remains the same, but only for installs and inner components. The mapping of installs to images may be arbitrary, or may only allow that installs from different servers are joined, but not that installs from one server are separated, or vice versa. Joining is typically easier than separating (because of the corresponding application data and helper software), but as we will see in the enterprise example below, certain levels of unstacking may be needed because current servers tend to contain more components than one is likely to want in the cloud images.

Software stacking or unstacking may also be permitted, i.e., we may permit that software components of the same type, which were so far inside different outer components, can be stacked together into one, or vice versa. In Figure 2, software stacking may allow that we move Module 3 into application server A, omit application server B, and thus make this application server install identical to those in multi-image templates M, M', and M''. If this is permitted, it may be conditioned on a node configuration compatibility test as described above. Unstacking may permit separation of software components that were previously together. For instance, we might move application servers A and B into different installs, and move URL w onto a different web server. Then we can split w and B off onto a new server S' (only for the matching algorithms, not onto a real physical server). We then get two independent connectivity components S, T and S', U that we can each map to multi-image template M.

Generally, in variants where the matching is not precise, the matching step may score different matchings according to the differences between source systems and multi-image templates, and possibly give cost estimates for each change. These scores or cost estimates are taken into account in selecting the overall mapping in Phase 3.

VI. Enterprise Example

In the following, we analyze an actual enterprise environment. The discovered source data are real, but the real use case was not yet a migration to a cloud with multi-image templates. Hence we analyze what a suitable set of multi-image templates for this environment could be, and what coverage we obtain by using different matching criteria. Thus we use our matching algorithms with more free variables: In each step, the relevant characteristics of the multi-image templates are initially unknown. We simultaneously choose these characteristics and the matching criteria, aiming at a good tradeoff between degree of standardization and how many source servers pass this matching step.

The environment contains about 1600 servers. As operating system (OS), more than 1000 run AIX, 200 Linux, and 300 Windows, plus a few outliers. Note that this is not an example of an entire enterprise, but already an environment preselected for a potential transformation. This allowed us to gather detailed software data. We used the Galapagos tool [11] for the discovery.

As Step i of the preparation phase, we excluded, e.g., compilers, interpreters and IT management software from our discovery results. The remaining key middleware installations are shown Table I. 1204 of the servers contain at least one such key middleware installation. We anonymized the actual products by naming only classes, with numbers for different products. E.g., DBMS1 means a first database product, DBMS2 a second one, WEB means web servers, PROX web proxy servers, APS application servers, MESS messaging software (queues), CRM customer relationship management systems, COLL collaboration products, and SCHED workload schedulers. The numbers per OS do not quite add up because of the few outlier OSes.

Software	#hosts	#installs	#AIX	#Linux	#Windows
PROX1	7	12	7	0	0
WEB1	304	465	268	29	6
WEB2	86	86	0	0	86
APS1	289	324	238	40	8
APS2	3	3	0	1	2
DBMS1	818	1547	633	53	129
DBMS2	110	110	5	92	13
DBMS3	68	91	0	0	68
DBMS4	45	60	1	26	16
DBMS5	28	28	1	27	0
MESS1	150	150	132	17	1
MESS2	3	3	0	0	3
COLL1	214	310	170	7	36
CRM1	31	31	31	0	0
SCHED1	78	83	74	0	4

TABLE I
SOFTWARE INSTALLS IN OUR ENVIRONMENT

As to the choice of software for the multi-image templates, there is a clear dominance of WEB1, APS1, DBMS1, MESS1, and COLL1 in this environment. They are particularly dominant on the AIX servers, while the Linux servers in particular contain a variety of other databases, and the Windows servers also contain a lot of Windows-specific web servers and databases. Given the dominance of APS1 as an application server, almost all the 3-tier structures, which we mainly want to migrate to instances of multi-image cloud templates, contain APS1 as their middle layer. Slightly looking ahead in the algorithm, we analyzed the dependencies of APS1 components. They show that APS1 is predominantly used with WEB1, DBMS1, and MESS1. Hence for this example, we decide that our multi-image template library focusses on these four software types.

In Step ii, we use a loose matching criterion that allows all versions of these software types. This criterion is reasonable as an enterprise that standardizes on multi-image templates is likely to also standardize on the newest software versions. Upgrades (unless from very old versions) are relatively cheap compared with other modifications we have discussed. Thus Step ii excludes all servers that contain any major software install outside these four, or none of the major software types at all. We ignored the OS types in this step, assuming that templates for the 3 major OS versions would be made and upgrades would be allowed. OS upgrades are significant work, but given the templates, we have to perform per-application migration rather than simple physical-to-virtual transformation anyway. This leaves us with 683 servers that are potential candidates for migration to such a multi-image template library.

Step iii did not further restrict this, as no network-based discovery was run in this environment. This is not a choice we recommend.

In Step iv we encountered very many exclusions. This surprised us as we had performed the initial test that application servers of type APS1 are mainly used with the other software types we chose, and because this environment was chosen for a transformation project where one should expect that multi-component applications would be treated as a whole. Nevertheless, many of the remaining internal connectivity components (i.e., if one does not count dangling dependencies) have dangling dependencies, i.e., connections to servers that contain other software or that were not scanned so that one cannot decide about their suitability for a multi-image template. We see this for all our types of connections, but in particular for connections from database aliases to real databases. Hence in this enterprise IT environment, real database servers often seem outside the subsets of servers where discovery is performed at the same time. In classical types of migrations of business applications, this is indeed less of a problem, because outgoing dependencies to servers that are not currently migrated typically do not require reconfiguration on either side. If real migration to multi-image templates were considered for this environment, one should extend the discovery recursively to connected servers. Else one would need to keep the actual database servers off the multi-image templates, but that contradicts the idea of preconfiguring all the important dependencies on the template images.

As a matter of interest, we show the sizes of the internal connectivity components of the 683 candidate servers in Table 2.

Size	1	2	3	4	5	6	7	9	13	20	22	31	82
Count	381	13	13	6	2	2	2	1	1	1	1	1	1

TABLE II
CONNECTIVITY COMPONENT SIZES AND COUNTS AMONG SERVERS WITH ONLY THE SOFTWARE CHOSEN FOR TEMPLATES.

However, after Step iv we only found ten truely independent connectivity components, five of size 2 and five of size 3. We studied their reduced version according to Step v. In principle, we are now carrying out Phases 2b and 2c, image and dependency matching, with the template structures as free variables. Given this small set, it amounted to analyzing the 10 given graphs for similarities. A few only have a database and one or two remote versions of it. The rest are all different. We describe those here as samples of how even small structures of 3-tier software types can vary in a real enterprise.

The simplest two-server structure is one server with APS1 and a remote DBMS1, and the corresponding DBMS1 server. Another is two servers, each with a connected 3-tier structure of WEB1, APS1, and DBMS1, where the two web servers interact in both directions, see Figure 6 left. The internal structure of the two application servers and the two database servers is similar, but with different data; they do indeed seem to belong together. Another structure is two servers with WEB1 and APS1 each, and with one-directional communication between the two web servers.

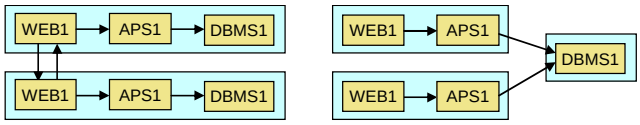

Fig. 6. Two of the real source connectivity components.

The first 3-server structure has two servers with WEB1 and APS1 each that seem to be replica, both accessing the same DBMS1 on a third server, see Figure 6 right. The next structure contains one server with WEB1, APS1, and DBMS1, and two other servers with remote database definitions to two different databases on the first server. The last one has two APS1 servers, each accessing a different database in the same service on a third server, and one of the first two servers also has a remote database definition to yet another database on the third server. The two APS1 servers have similar modules, but one contains additional management modules.

As this was a very small sample, we also show some statistics over larger server sets from earlier steps that may be useful to take into account in designing multi-image template sets, or in research and development of efficient methods for complex migrations.

The sum of the column #installs in Table I is 3304, i.e., on average 2.75 major software installs per server. Hence one either needs to consider template images with multiple major software types too, or, to reduce the image sprawl, explore unstacking options to obtain simpler and fewer templates. Among the 683 candidate servers after Step ii, out of the 15 possible combinations of the four software types on the images, only one does not occur (APS1+MESS1). The distribution is 311 servers with 1 software type, 182 with two, 105 with 3, and 85 with 4. Hence if we decide on template images with only 1 main software type, we almost halve our candidates, or need unstacking. Else we get quite a variety of individual images even among just these 4 software types.

We also saw a significant number of servers with several installs of the same software. A first conjecture may be that

they are successive product versions and the older ones may no longer have services running. For the former, deletion may be helpful for standardization, but this can typically not be decided automatically, but needs validation with server owners and business application owners, as there may be cases where services are only instantiated under certain conditions, even though one tends to think of production environments as having their 3-tier services constantly running. Actually, for about 1/3 of our installs we found no service.

VII. Related Work

Clouds and their potential benefits are described, e.g., in [1, 3, 15]. We assume that readers are familiar with commercial offerings. Migration of existing business applications into clouds, however, is not yet common in large enterprises. Typically large enterprises use clouds for development and test environments where content is newly assembled on a cloud image after deployment, or in a few cases for applications that are new or have to be reprogrammed anyway.

Multi-image templates have initially been proposed as a software deployment mechanism in [2, 7, 8], without actual prebuilt images in an image library. The use of multi-image templates in clouds has been specifically addressed in [5]. This means the preparation of actual groups of virtual images according to the templates, and the use of deployment scripts to adapt the communication setups among the images to actual addresses when they are instantiated on actual servers.

Migration of business applications is a serious topic in industry, but rarely published. Classic use cases are hardware refresh, server consolidation, operating system upgrades, software upgrades, software consolidation (e.g., vendor changes) and software stacking (e.g., multiple databases into a database farm). All this is done on a larger scale than, e.g., SOA transformation or business process transformation. An early industrial white paper is [16]; a more recent overview is [18]. Configuration migration and changes are addressed in [13, 17]. An application of such techniques to clouds is described in [20]. This kind of automation does not enable any significant changes yet. Hence matching of existing stuctures with very similar cloud templates, as we investigate it, is important for finding candidates for cost-effective migration.

Placement of virtual machines on servers is addressed in [14, 19]; with the advent of clouds with live migration [6] this is evolving from a migration topic into a cloud-internal management topic. These works are about the hardware consolidation aspects of virtualization, while ours is about the software standardization aspect. A work on network aspects of cloud migration is [10]. It considers network and firewall settings if not all servers are migrated to a cloud. It assumes pure P2V (physical-to-virtual) transformation without software standardization and thus without any matching with given templates.

A start into analyzing servers for migratability to standard images was made in [9]. It considers single images and installs only, and thus no tree or graph matching. Other novel features in our work are the variable matching criteria, the optimization of the selection of the template set, and a real enterprise example. We consider real examples very important; e.g., they made the assumption that each software type occurs at most once on each source server, which we found to be often violated.

VIII. Conclusions

Standardization of virtual images via clouds is considered a key factor in reducing IT management cost, the dominant cost of current IT. Virtualization is a key enabler of this option, but, if not used carefully, can also cause the opposite effect, as image sprawl may increase overall IT management costs. As most business applications need more than one image, there are efforts to provide multi-image templates in catalogs and actual multi-image structures ready for deployment in clouds.

We have provided a framework for planning how to migrate existing business applications to such clouds, as this is the only way of gaining wide adoption and significantly decreasing real enterprise costs in a reasonable period of time. We have presented matching criteria that enable a tradeoff between the difficulty of migration and coverage, i.e., how much of the existing software and data can be migrated. We have seen in an enterprise environment with 1600 servers that the migration will not be trivial, because the current structures are heterogeneous and complex. In particular, we have seen that options to unstack currently co-located software will be important. The complexity of the example supports the overall hypothesis that large gains can be obtained if standardization to a cloud with multi-image templates is actually performed.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commununications of the ACM (CACM)*, 53(4):50–58, 2010.

[2] W. Arnold, T. Eilam, M. H. Kalantar, A. V. Konstantinou, and A. Totok. Automatic realization of soa deployment patterns in distributed environments. In *Proc. 6th Intern. Conf. Service-Oriented Computing (ICSOC), LNCS 5364*, pages 162–179, 2008.

[3] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2009.

[4] A. Caracas, D. Dechouniotis, S. Fussenegger, D. Gantenbein, and A. Kind. Mining semantic relations using NetFlow. In *3rd IEEE/IFIP Int. Workshop on Business-Driven IT Management (BDIM)*, pages 110–111, 2008.

[5] T. C. Chieu, A. Mohindra, A. Karve, and A. Segal. Solution-based deployment of complex application services on a cloud. In *IEEE Intern. Conf. Service Operations, Logistics and Informatics (SOLI)*, pages 282–287, 2010.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2005.

[7] T. Eilam, M. H. Kalantar, A. V. Konstantinou, and G. Pacifici. Reducing the complexity of application deployment in large data centers. In *Integrated Network Management (IM)*, pages 221–234, 2005.

[8] T. Eilam, M. H. Kalantar, A. V. Konstantinou, G. Pacifici, J. Pershing, and A. Agrawal. Managing the configuration complexity of distributed applications in internet data centers. *IEEE Communications Magazine*, 44(3):166–177, 2006.

[9] R. Filepp, L. Shwartz, C. Ward, R. Kearney, K. Cheng, C. Young, and Y. Ghosheh. Image selection as a service for cloud computing environments. In *Proc. 8th Intern. Conf. Service-Oriented Computing (ICSOC), LNCS 6470, to appear*, 2010.

[10] M. Y. Hajjat, X. Sun, Y.-W. E. Sung, D. A. Maltz, S. G. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. In *Proc. of ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 243–254, 2010.

[11] N. Joukov, B. Pfitzmann, H. V. Ramasamy, and M. V. Devarakonda. Application-storage discovery. In *3rd Annual Haifa Experimental Systems Conference (SYSTOR'10)*, Haifa, Israel, May 2010. ACM.

[12] N. Joukov, V. Tarasov, J. Ossher, B. Pfitzmann, S. Chicherin, M. Pistoia, and T. Tateishi. Static discovery and remediation of code-embedded resource dependencies. In *12th IFIP/IEEE Intern. Symp. on Integrated Network Management (IM'2011), to appear*, 2011.

[13] Q. Ma, Y. Li, K. Sun, and L. Liu. Model-based dependency management for migrating service hosting environment. In *Proc. IEEE Intern. Conf. on Services Computing (SCC 2007)*, pages 356–363, 2007.

[14] S. Mehta and A. Neogi. ReCon: A tool to recommend dynamic server consolidation in multi-cluster data centers. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 363–370, 2008.

[15] D. Owens. Securing elasticity in the cloud. *Commununications of the ACM (CACM)*, 53(6):46–51, 2010.

[16] Sector7. An application centric view of server consolidation and IT optimization. Sector7, 2003. http://www.sector7.com/products_and_services/openvms/server-consolidation-it-optimization-whitepaper.pdf.

[17] M. Sethi, K. Kannan, N. Sachindran, and M. Gupta. Rapid deployment of SOA solutions via automated image replication and reconfiguration. In *Proc. IEEE Intern. Conf. on Services Computing (SCC 2008)*, volume 1, pages 155–162, 2008.

[18] M. Torchiano, M. D. Penta, F. Ricca, A. D. Lucia, and F. Lanubile. Software migration projects in Italian industry: Preliminary results from a state of the practice survey. In *23rd IEEE/ACM Intern. Conf. on Automated Software Engineering - Workshop Proceedings (ASE Workshops)*, pages 35–42, 2008.

[19] A. Verma, P. Ahuja, and A. Neogi. pMapper: Power and migration cost aware application placement in virtualized systems. In *Middleware 2008*, pages 243–264, 2008.

[20] C. Ward, N. Aravamudan, K. Bhattacharya, K. Cheng, R. Filepp, R. Kearney, B. Peterson, L. Shwartz, and C. C. Young. Workload migration into clouds – challenges, experiences, opportunities. In *IEEE 3rd Intern. Conf. on Cloud Computing*, pages 164–171, 2010.

[21] X. Zheng, M. Zhan, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *Proc. 8th Symp. on Operating Systems Design and Implementation (OSDI 2008)*, pages 117–130, San Diego, CA, December 2008.