Migration to Multi-Image Cloud Templates

Birgit Pfitzmann, Nikolai Joukov
IBM T. J. Watson Research Center, Hawthorne, NY USA
{bpfitzmann,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 typically contain multiple images with dependencies, 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 handles approval workflows, and a database that stores the expense reports. 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.

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.

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. This fact 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. Nevertheless, the base setup of the dependencies can be done at the service level and thus in the preconfigured images.

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.

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.

![Multi-image template M](image)

![Multi-image template M'](image)

Fig. 1. Example multi-image templates.

Multi-image template $M'\text{' is parameterized, i.e., there are parameters, }i\text{ and }j\text{, that a user can select when choosing this template from the catalog. At the front end, the multi-image template }M'\text{ includes an image }I'\text{' 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'\text{, e.g., if a user needs ten web servers and thus chooses }i=10\text{, then image }I'\text{ 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\text{ replicated databases. Instead of the keyword “}i\times\text{ times same”, there may be “}i\times\text{ times different”, meaning that }i\text{ independent instances of the image will be deployed.}

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. 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 $P$.

![Diagram](image)

**Fig. 2.** Example source servers.

**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.

**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. 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, 10, 11, 20]. 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.

In Phase 2, potential individual multi-image template matches are found. 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. 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.
B. 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.

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).

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 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 match with varying strictness, e.g., only software components with exactly the same version match, or source versions match newer minor image versions or all newer image versions, or even similar products of different vendors match. This has consequences on the migration costs. Each matching criterion defines a relation “≤” over the type set that is required 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 name labels, as user-given names seem too unlikely to ever match names in templates.

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.

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. 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.

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 match with varying strictness, e.g., only software components with exactly the same version match, or source versions match newer minor image versions or all newer image versions, or even similar products of different vendors match. This has consequences on the migration costs. Each matching criterion defines a relation “≤” over the type set that is required 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 name labels, as user-given names seem too unlikely to ever match names in templates.

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.

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. 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.

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 ∉ SI. Further exclude source servers without any install s ∈ SI.

In Step iii, we exclude source servers that have a dependency on the server level that has not been associated.
I. Delete helper software, infrastructure dependencies, and client connections

ii. Exclude servers with OS or installs not matching any templates, or without any template installs

iii. Exclude servers with un-understood dependencies

iv. Exclude connectivity components with dangling dependencies

v. Reduce source graph to inclusion depth of templates, lifting the dependencies

vi. Match connectivity component sizes

Fig. 4. Preparation phase for precise matching.

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 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 \( P, R, X \). 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.

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.

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 considered file systems and servers as infrastructure, and incoming dependencies on web server URLs as irrelevant.

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 \( \Phi \) 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 graph unification with commutative operators. We present pseudo-code of such an algorithm \texttt{treematch} in the report version of this paper. Here we have to omit it for space reasons. (We hope the report propagates to \( \text{http://domino.research.ibm.com/library/cyberdig.nsf/index.html} \) soon. Else please ask authors via PC chair.)

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). For space reasons, the detailed description is only in the report version, see above.

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. In such cases, we can compute modification costs for choosing a multi-image template \( M \) with node mapping \( \lambda \) from Phases 2b and 2c as

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

where \( n \) are nodes in the multi-image template \( M \) and \( \text{mod\_cost}(x, y) \) 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
\[ \sum_{i=1}^{\sigma} \text{cost}(SC_i, M_i) + |\{M_i | i = 1, \ldots, \sigma\}|, \]
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.

V. OTHER MATCHING VARIANTS

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

A. Precise Matching with Parameterized Templates

For parametrized multi-image templates like $M'$ in Figure 1, the preparation phase is as before except for Step vi, and node matching exactly as before. For dependency matching, we produce an expanded template with the necessary concrete parameters, and then match the dependencies as before.

B. Less Strict Matching Variants

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

Unused dependencies in the multi-image templates may be allowed. In examples like $M$ with very few dependencies this makes little sense, but it could with multi-image templates that have a large variety of preconfigured dependencies.

Unused components in the multi-image template may also be permitted. In examples like $M$ with few software components per image this again makes little sense, but it can be important if one also lists minor software components such as compilers and infrastructure management software.

Additional dependencies on the source systems may be permitted. We do not recommend it, though, as a key goal of simplified management via multi-image templates is that dependencies are managed according to the templates.

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. E.g., as unstacking in Figure 2, the web server and the application server install from server $S$ may be split, thus making that part similar to images $I'$ and $K'$ of multi-image template $M'$. The algorithms $treematch$ and the dependency matching remain the same, but are called on installs instead of images. While stacking is typically easier, certain levels of unstacking may be needed in real enterprises because current servers tend to contain more diverse component sets than one wants 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.

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. We used the Galapagos tool [10] 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.

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

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. 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. 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 candidate servers.

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.

However, after Step iv we only found ten truly 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.

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.

We also made some relevant statistics over the larger set of servers. Multiple major installs per servers are common, on average 2.75 in Table I. Furthermore, even among our four chosen software types, 15 combinations occur on individual images. Hence if we can complete the graphs for this larger server set, we already know we will either get significant image sprawl, or significant exclusions, or we need to work on unstacking migration.

VII. Related Work

Clouds and their potential benefits are described, e.g., in [1, 3, 14]. 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

<table>
<thead>
<tr>
<th>Size</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>9</th>
<th>13</th>
<th>20</th>
<th>22</th>
<th>31</th>
<th>82</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>381</td>
<td>13</td>
<td>13</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table II: Connectivity Component Sizes and Counts Among Servers with Only the Software Chosen for Templates.
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, 6, 7], without actual prebuilt images in an image library. The use of multi-image templates in clouds has been specifically addressed in [5].

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 [15]; a more recent overview is [17]. Configuration migration and changes are addressed in [12, 16]. An application of such techniques to clouds is described in [19]. This kind of automation does not enable any significant changes yet. Hence matching of existing structures 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 [13, 18]; with the advent of clouds with live migration 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 [9]. 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 [8]. 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. 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