Automated Business Application Discovery

Michael Nidd
IBM Research, Zurich Lab
mni@zurich.ibm.com

Kun Bai, Jinho Hwang and Maja Vukovic
IBM Research, Watson Lab
{kunbai, jinho, maja}@us.ibm.com

Michael Tacci
IBM Global Technology Services
mtacci@us.ibm.com

***Abstract*—When planning a data center migration it is critical to discover the client's business applications and on which devices (server, storage and appliances) those applications are deployed in the infrastructure. It is also important to understand the dependencies the applications have on the infrastructure, on other applications, and in some cases on systems external to the client.**

Clients can only rarely provide that information in a complete and accurate manner. The usual approach then has been to obtain the information by asking the client's application and platform owners a series of questions but in most cases clients do not have the tools or skills to acquire the requested information. The lack of accurate information leads to project delays, increased cost and higher levels of risk.

In this paper we present an algorithm and tools for programmatically identifying and locating business application instances in an infrastructure, based on weighted similarity metric. We discuss results from our preliminary evaluation and the correctness of the algorithm. Such automated approach to application discovery significantly helps clients to achieve their project objectives and timeline without imposing additional work on the application and platform owners.

I. Introduction

Driven by the promises of access to elastic computing resources enterprises are keen to migrate migrate on-premise computing resources into the Cloud. It is not only the advantage of capital and operational cost reduction that is appealing, but also Cloud support for variety of deployment architectures via public/private/hybrid infrastructure-as-a-service (IaaS) offerings to meet different security and resource demands. Among others, the success story of Netflix open source software (OSS)− with hundreds of mid-tier services and applications, billions of requests per day, up to 70 billion events per day, thousands of EC2 instances in multiple regions−has been enticing enterprises to consider how Cloud can be leveraged to run their businesses.

Despite the significant interests, migrating enterprise-scale workloads has been technically challenging [1]. A deterrent preventing the rapid exodus to Cloud is the lack of holistic migration techniques and migration planning tools that can help minimizing business impact during migration. Today Cloud providers offer a per-unit based migration method only−P2V, V2V and P2P converters[1]−so that customers hardly dare to take on the whole responsibility to migrate thousands of on-premise servers to Cloud. Also, given a large number of servers running on different platforms, different system properties of source machines need to be considered to discover business application groups that can reflect realistic business impact, in turn create well defined migration wave plans (minimizing business impact). Migrating these business application groups together is critical to minimizing the business impact during migration to Cloud [2], [3].

Designing migration waves to allow for groups of servers that work together (business application groups) has been shown to be an important component of enterprise-scale migration [1], [4], [5]. Grouping servers reduces service disruption, in turn reducing migration costs, during/after migration waves. With well defined server groups, each group has high intra dependency with similar platform/application types, and low inter dependency between the groups. A pattern matching technique has been leveraged to search for certain profiles amongst the servers from the collected source site data, and compose server groups. System properties (i.e., server configurations and network connections) are used to represent the strength of the servers, helping grouping algorithms to find the best groups [2].

In this paper we take a step towards finding the best practical business application groups, by categorizing the system properties and communication patterns of each server, and then assigning weights to particular sections of those properties. Through this weighting process we identify similarity groups, for which signatures are calculated and used to identify further candidates. To validate the correctness of server groups, we compare the proposed server groups and correct server groups derived from interviews of practitioners and direct system evaluation by migration wave architects.

II. Related Work

Server grouping falls in the pattern matching domain, as we are looking for certain profiles amongst the servers in the source site. Spectral clustering represents a set of techniques which rely on the eigen- structure of a similarity matrix to partition points into disjoint clusters with points in the same cluster having high similarity and points in different clusters having low similarity. Spectral clustering and path-based clustering are related clustering approaches in the domain of machine learning and pattern recognition. They have demonstrated great performance on some clustering tasks, which involve highly non-linear and elongated clusters, as well as compact clusters. Chang et al. [6], propose a path-based algorithm for spectral cluster generation, defining a robust similarity measure that reduces the effects of noise and outliers in the data. Key challenges with spectral clustering are that one needs to (1) know how many clusters one is looking for, (2) be able to ignore the outliers that are in no group at all, and (3) tolerate multiplicity for elements that may be in more than one group.

[1]P and V stand for physical and virtual, respectively. For instance, VMware provides a converter to migrate into VMware environments.

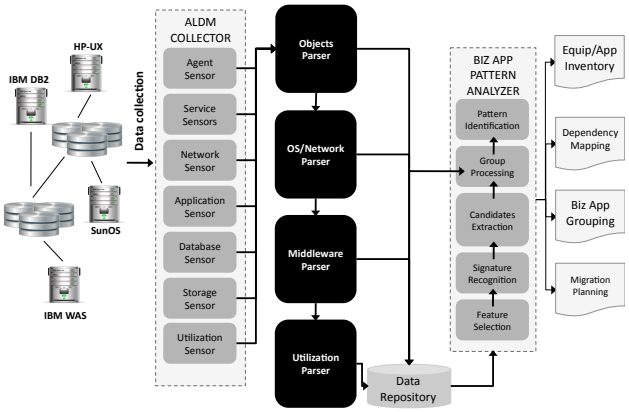

Fig. 1. ALDM Discovery Framework

Migration work [2], [3] necessarily considers grouping of servers. Al Kiswany et al. [2] group servers by OS, OS distribution or their relationship to a specific application. Keller et al. [3] propose grouping of components in order to devise the best migration approach to use for that group, the migration order across groups, and optimizing migration based on appropriate network measurements

III. Collecting System Descriptions

Our implementation builds on Analytics for Logical Dependency Mapping (ALDM) [7], an existing IBM infrastructure discovery framework, as shown in Figure 1. The front-end is a light-weight discovery kit that runs in customer data centers. All of the data is collected at defined intervals (e.g., 15 minutes) over a period of one or two weeks. information collected includes middleware configuration, server CPU, memory, and I/O utilization, network traffic activities, and running processes. The back-end is our data processing engine running in the IBM SoftLayer Cloud, which processes and analyzes the collected data. It provides two categories of data for further analysis: static and dynamic. Static tells us basic information about server setting, such as CPU capacity, IP Addresses, Operating System, and other hardware related information. Dynamic data consists of a list of open ports, traffic flow, running applications, CPU, memory, I/O usage trending over time, server to server and application to server dependencies.

The intermediate results are stored as an XML file. Further analysis, such as Business Application mining and interactive visualization use this file as their input. The ALDM processing engine generates a highly accurate inventory report, business application signatures, graphic representations of dependencies, and other reports that inform migration engineers during the design of affinity groups and migration plans.

IV. Measuring Similiarity

Based on the scan data, what can we say about the similarity between two systems? ALDM has a tool for comparing two XML system descriptions, although its normal use is to compare two images of the same system. Our challenge here was use comparisons between the dscriptions of different systems to deduce functional similarity.

The second challenge is to use this measure of similarity to propose groups of servers that may comprise business application groups. Algorithms for proposing groups will be discussed further in V, but expressing similarity as scalar "distance metrics" between pairs of systems would be the least restrictive starting point for that next step.

A. Comparing System Descriptions

The comparison detects insertions, deletions, and changes. Elements are considered equal if they have the same qualified name (XML tag), and the same attribute values. Changes to particular attributes that are assigned arbitrary values during scanning are ignored. Insertions and deletions correspond to elements in either of the system descriptions for which no equal element exists in the other. Changes are elements that are equal, but contain insertions or deletions among their child elements.

Any text in the server description that matches (ignoring case) the server's own name is converted to a marker that is considered to match similar markers from other descriptions. Although a self-reference may be using a locally provided service that is also used by other servers in the same group, it more commonly corresponds to self-references in the configurations of other group members. These are frequently found in URLs (e.g. a local service) or in file paths (common in WebSphere® Application Server installations).

The simplest way to derive a scalar similarity metric from the difference between two system descriptions is just to count the elements they have in common. If system A is described with 123 elements, System B with 58 elements, and 22 elements are common to both sets, then that's 44 of 181 elements in total, or 24% similar.

Unfortunately, not all lines of the system description are equally useful when determining similarity. For example, one section records the version of ALDM that was used to generate the scan. A more reasonable comparison might ignore this group of elements, and possibly give more value to the installed software packages and network connectivity than it gives to disk usage and network interface cards.

We applied domain experience to assign weights to some sections of the system description, using these as multipliers for shared elements contained in these sections. The sub-tree evaluations were measured as value pairs ($number\_in\_common \times weight$, and $total\_lines \times weight$) to allow sub-trees of different weights and sizes to be combined reasonably, as shown in Figure 2. This allows a large number of similar values in a low-weight branch to be as important as a small number of values in a high-weight branch.

Most elements have a weight of one, but other weights have been assigned to elements by name, based on expert opinions. The lowest weight used is zero, such as for elements that describe the scan scripts (version, run date, etc.) rather than the system itself, while weights as high as five are used for network connectivity sections.

Weighting greatly reduced the number of systems that appeared very similar, as shown in Figure 3. This gives a

```
Given: float myWeight
Given: boolean myAttributesChanged
Given: List childComponents
int p= myAttributesChanged ? 0 : 1;
int t= 1;
for all (WeightedSimilarity c in childComponents) {
    int w= c.getWeight();
    p += c.getMatchingElementCount() * w;
    t += c.getTotalElementCount() * w;
}
myMatchingElementCount= p;
myTotalElementCount= t;
```

Fig. 2. Calculating the weighted similarity value for an element.

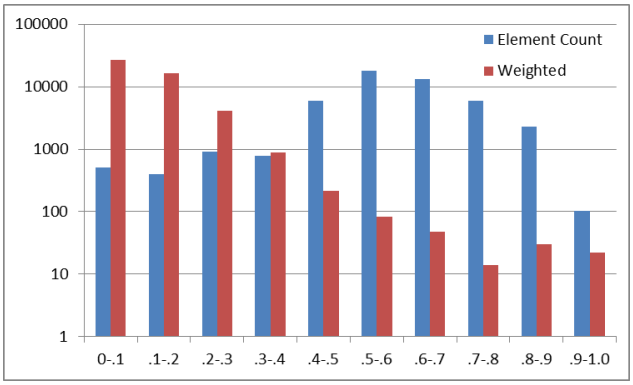

Fig. 3. Comparing the distribution of similarity values using lines in common with weighted similarity in a data set of 220 systems; weighted comparison is much more selective.

distribution that more intuitively matches what one would hope for in generating groups of systems.

V. Proposing Groups

Efficient clustering algorithms, such as spectral clustering[8], can find clusters in data from pairwise "distance" metrics such as we have here. Several factors make these algorithms inappropriate for this application:

1) Some elements are not in any group.
2) Some elements are in more than one group.
3) We have no good guesses for how many groups should be found.

An alternative approach is to first select thresholds for "similar," or "not similar;" then to run a greedy algorithm to propose possible groups:

1) Assign each server an ordinal reference and create a single-element group containing that reference.
2) For each group under evaluation, look for servers with higher ordinal values than all members that are within the similarity threshold to all members.
3) For each match, create a new group containing the superset including the new server.
4) Repeat 2-3 until all groups have been considered against all higher-reference servers.
5) Remove any proper subgroups.

If similarity is defined too generously, the memory required for this algorithm will explode, so it is uesful to include a trigger that re-starts with higher thresholds if the number of possible groups grows too large.

A. Considering Communication

Servers collaborating to provide a single application need to communicate. This fact has been used to augment the similarity metric in proposing server groups. Step 2 in the algorithm discussed above can be modified to adjust the similarity threshold based on other criteria.

The following three communication patterns appear most significant to determining the likelihood of System A and System B being members of an application group:

- System A opens communication channels with System B (or vice verca).
- System A and System B both open communication channels with each other.
- Some other system opens communication with both System A and B.

As with different areas in the system description, we have weighted these different communication dependencies. One-way connections are ignored, pairs with mutually-initiated connections are always considered similar, and a lowered (but still non-zero) threshold for similarity is used for pairs of systems that both receive communication from the same external system.

B. Application Signatures

The same tool that is able to compare two XML server definitions in ALDM system scans is also able to generate a "signature" for any two or more server definitions. This signature contains the elements in common among all components (including the self-reference tokens) as discussed in Section IV-A.

Once a partial group is known, possibly as the result of an interview with the business owner of an application, these signatures can be useful in finding other servers that were not mentioned. One common cause for such omissions is to list the production servers, and to forget to mention the development and test servers.

This same signature technique has proven useful in extending groups proposed by similarity-based clustering. Signatures return all servers that match the elements shared among the specified components, overlooking the possibly large number of elements that are not shared. This can help to cut through distracting "noise" in some server descriptions, such as may be caused by a large number of middle-ware products being installed on one group member only.

VI. Evaluating Correctness

The process to compare discovered Business Application (BizApp) associations is done in two steps. The associations or groupings of systems identified by the BizApp discovery method are first compared to the initial inputs provided by the business application owner or customer. The initial inputs

consist of a business application name or ID and a list of known server nodes that provide a service relevant to the functionality of the application. This is called the app to server mapping. Any deltas in the server list of this mapping are marked for reconciliation in the second step of the validation.

The second step includes an interview session with the business application owners. The goal of the interview is to gather a basic functional understanding of the BizApp, a classification of the service provided by each of the servers (i.e. web. Database, messaging, etc.), an environment designation such as production, and any component details. Component details may include middleware applications, database names, services and ports utilized by the BizApp. Once the data elements are captured the information is input and an updated grouping is produced. The updated grouping will also include any inferred dependencies that were deemed relevant by the application owner during the interview. In contrast any dependencies not relevant are filtered and absent in the updated grouping. The final grouping, services and associated data stores is then verified by the business application owner(s).

For our testing of this new system, we used data from a complete project. This means that the first step could be carried out with unusually good precision, since we had already iterated on this process with manual categorization. It also meant that results evaluation was not evaluated in fresh interviews, as we would prefer, but was compared with the results of an extensive manual review.

In our set, we knew of about 300 servers supporting 35 multi-server applications. We had detailed scan data for 159 of those servers, representing exactly two members of 20 of the known applications, and three or more members of another 20. 58 groups were suggested by our system. If we define a correct group proposal as one which successfully identifies at least two members of a genuine group, then the results break down as follows:

Groups of 2:	21 false, 5 correct (App sizes: 2×5, 1×3, 2×2)
Groups of 3:	12 false, 4 correct (App sizes: 1×3, 3×2)
Groups of 4:	4 false, 2 correct (App sizes: 1×5, 1×16)
Groups of 5+:	2 false, 8 correct (all correct contained 6 or more servers from a group of 16)

For our 159 detailed scans, a manual count of the number of independent pairs that belong to a single application found 347. From the $\binom{159}{2} = 12561$ pairs possible, let $C = 2.8\%$ represent the chance of a random pair being "correct". The chance of groups larger than 2 being correct should be calculated without replacement (because a group of n distinct members will always have $\binom{n}{2}$ *distinct* pairs), but for small groups this will be almost the same number, so for groups of size 3 and 4 the probability of random success is approximated by one minus the chance of $\binom{n}{2}$ independent failures:

$$p_n \approx 1 - (1 - C)^{\binom{n}{2}}, \qquad n \ll 159$$

This calculation leaves the chances of random success for different group sizes $p_2 = 2.8\%$, $p_3 = 8.1\%$, and $p_4 = 15.5\%$. The results were 19%, 25%, and 33% respectively, which is significant evidence, especially considering that one group of four exceeded the definition of "correct" by finding four out of five systems in an actual application; similarly, one of the groups of three represented all three members of a single application.

VII. Results and Conclusions

The challenge for this project was to propose groups of servers that are likely to be working together to deliver any single business application. We have shown that our weighted comparison is a reasonable nearness metric for determining likely partners. We have further shown that the groups determined with this metric have a tolerable rate of false positives in our customer data sets for groups larger than two, meaning that these false positives will not create significant unnecessary restrictions on the servers that can be assigned to groups during wave planning. Thus, this technique will prevent some business application sets from being split accross multiple waves without otherwise interfering with the wave planning process.

The run times have been on the order of minutes for our data so far, which extends to hundreds of servers. In the year ahead, we will be testing on larger engagements, which are expected to scale with the square of the number of servers. We anticipate overnight runtimes for these larger engagements, but consider this acceptable for the first step in a planning process that lasts weeks anyhow.

Trademarks

References

[1] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 243–254, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1851275.1851212

[2] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: Virtual machine co-migration for the cloud," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, ser. HPDC '11. New York, NY, USA: ACM, 2011, pp. 159–170. [Online]. Available: http://doi.acm.org/10.1145/1996130.1996153

[3] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live migration of an entire network (and its hosts)," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI. New York, NY, USA: ACM, 2012, pp. 109–114. [Online]. Available: http://doi.acm.org/10.1145/2390231.2390250

[4] K. Bai, N. Ge, H. Jamjoom, E.-E. Jan, L. Renganarayana, and X. Zhang, "What to discover before migrating to the cloud," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, May 2013, pp. 320–327.

[5] J. Zhang, L. Renganarayana, X. Zhang, N. Ge, V. Bala, T. Xu, and Y. Zhou, "Encore: Exploiting system environment and correlation information for misconfiguration detection," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 687–700. [Online]. Available: http://doi.acm.org/10.1145/2541940.2541983

[6] C. Hong and D.-Y. Yeung, "Robust path-based spectral clustering," in *Pattern Recognition*, 2008, pp. 191–203.

[7] "IT Infrastructure Discovery, Analytics for Logical Dependency Mapping (ALDM)," http://www.ibm.com/services/ALDM/.

[8] A. Y. Ng, M. I. Jordan, Y. Weiss *et al.*, "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing systems*, vol. 2, pp. 849–856, 2002.