What to Discover Before Migrating to the Cloud

Kun Bai, Niyu Ge, Hani Jamjoom, Ea-Ee Jan, Lakshmi Renganarayana, Xiaolan Zhang
IBM T. J. Watson Research Center
19 Skyline Dr., Hawthorne NY 10523
{kunbai, niyuge, jamjoom, ejan, lrengan, cxzhang}@us.ibm.com

***Abstract*—The adoption of the cloud computing model continues to be dominated by startups seeking to build new applications that can take advantage of the cloud's pay-as-you-go pricing and resource elasticity. In contrast, large enterprises have been slow to adopt the cloud model, partly because migrating legacy applications to the cloud is technically non-trivial and economically prohibitive. Both challenges arise, in part, from the difficulty in discovering the complex dependencies that these legacy applications have on the underlying IT environment. In this paper, we introduce a novel Kullback-Leibler (KL) divergence based method that can systematically discover the complex server-to-server and application-to-server relationships. We evaluate our method using five real datasets from large enterprise migration efforts. Our results demonstrate that our new method is capable of finding critical application correlations; it performs better than traditional approaches, such as Bayesian or mutual information models. Additionally, by cleverly subdividing the sample space, we are able to uncover intriguing phenomena in different subspaces. These analyses aid migration engineers in a variety of tasks ranging from migration planning to failure mitigation, and can potentially lead to significant cost reduction in migration to cloud.**

I. Introduction

Cloud computing is disrupting the traditional models of delivering IT. Its appeal arises from features like *pay-as-you-go pricing*, *low upfront investment*, *resource elasticity* and *well-defined management APIs*. These "zero-friction" features have fueled rapid adoption by startups developing new applications. However, large enterprises—which are plagued with legacy applications—have been slow to adopt the cloud model. This is primarily because of the associated cost—both technical and economical—of migrating legacy applications to this new delivery model.

In theory, migrating an enterprise application to the cloud should be a straightforward task. It should require packing the application codebase and underlying data, then instantiating them on the target cloud environment. In practice, the migration process is often error-prone and high-risk for three primary challenges. First, enterprise applications have strong dependencies on the underlying operating systems (OSs), network topology and configurations, security policies, storage subsystem, etc. Figure 1 shows a typical dependency structure for an enterprise application. Second, these dependencies are seldom fully-known. Based on our experience with over 20 large enterprise migration efforts, even with efforts like CMDB [1], a significant percentage of application dependencies are either missing or incorrect. Third, enterprise applications are also mission-critical. Thus, detangling an application and migrating it cannot negatively impact its day-to-day operation.

Given the above challenges, there is clearly a strong need for enterprises to discover current deployments and establish complete mappings, starting from the enterprise application all the way through the software stack to server hardware, storage subsystem, and network. There has been extensive research work on dependency discovery across multiple research domains [2], [3], [4]. Network-based monitoring via NetFlow information from routers [3] and probing running applications from the network [4] are two well-known methods. There are also a number of commercial products focusing on IT discovery [5], [6]. They generally work by either statically exploring configuration files or monitoring the running system. Configuration-based approaches depend on the configuration files accurately capturing the dependencies. In contrast, monitoring-based approaches depend on frequent occurrences of specific events (e.g., establishing a TCP socket between two servers). When used together, these tools provide a good picture of a deployed system. However, with thousands of servers scanned and discovered, tremendous amount of information about internal (components in a server) and external (server-to-server) dependencies are captured. Combing and sorting out this information in a meaningful and insightful way is non-trivial and often error-prone.

In this paper, we propose a novel technique for systematically discovering complex server and application dependencies, and validate our findings using real enterprise data. Our work builds on an earlier system called Galapagos [2], which mainly focuses on automatically constructing applications-data relationships. In this paper, we use Galapagos as a data collection agent only. We then introduce, implement, and evaluate a novel and scalable Kullback-Leibler (KL) divergence-based method for discovering complex server-to-server and application-to-server relationships.

To summarize, this paper has the following technical contributions:

- We introduce a novel Kullback-Leibler (KL) divergence-based method that can systematically discover the complex server-to-server and application-to-server relationships. To the best of our knowledge, this is the first time such method is applied to the domain of server/application relationship discovery.
- We evaluated the methods using five real enterprise datasets that were collected during large migration projects. Our results demonstrate that the new method is capable of finding critical application correlations; it performs better than traditional methods (such as the Bayesian and mutual information models) for discovering server-to-server and application-to-server relationships.
- We perform additional dependency analysis by clev-

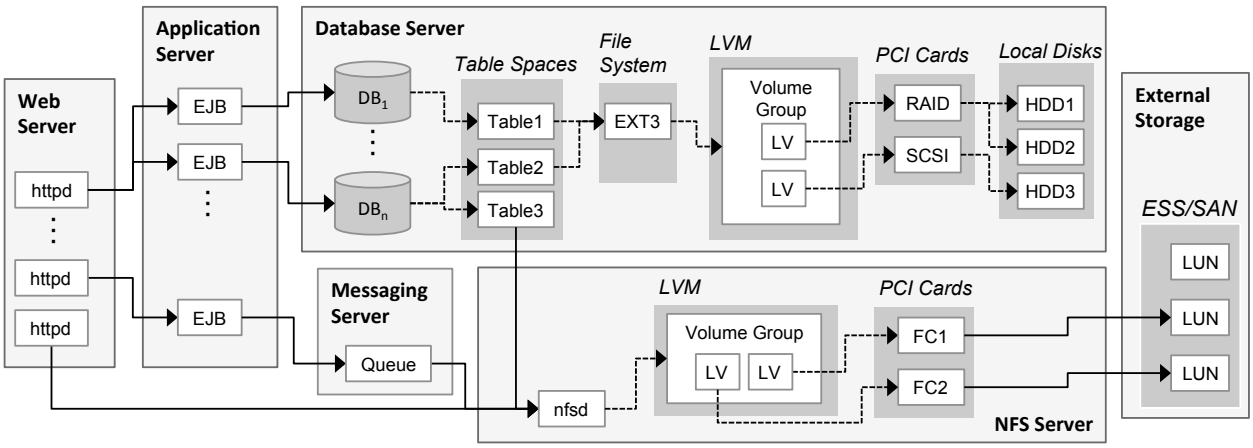

Fig. 1. A typical enterprise software and storage stack. An arrow from Component A to Component B means that A depends on B. A dashed line depicts internal—*server*—dependency and a solid like depicts external—*network*—dependency.

erly subdividing the sample space and show that we are able to uncover intriguing phenomena in different subspaces (grouping).

The rest of paper is organized as follows. In Section II, we describe the role of discovery in planning a large-scale migration. In Section III, we propose our probabilistic approaches to generate meaningful dependency and co-occurrence patterns. Three migration application scenarios are discussed in Section 4, followed by empirical evaluation using KL method. In Section 5, we discuss related works in cloud computing, existing techniques in dependency discovery. Finally, we conclude in Section 6.

II. Migration Planning

In this section, we describe the role of migration plans during a typical large migration project. At a high level, a migration plan consists of a series of phases, each detailing (1) what parts (cluster) of the IT infrastructure to move, (2) their target environment, and (3) the change window to perform the move. Creating a plan starts with discovering as much information about the IT infrastructure as possible. The goal is to minimize the overall cost of migration and reduce the risk of any potential down time.

A migration plan is needed because it is infeasible to migrate tens of thousands servers—typical of enterprise customers—in a single shot. Thus, a migration engineer needs to partition the servers into smaller clusters, each of which can then be migrated during a small change window (e.g., during the weekend). Creating a migration plan is an extremely complicated task. In many ways, it is as a large multi-constraint scheduling problem. At minimum, the plan needs to satisfy the following two constraints:

- *Each batch should be small enough so that the servers can all be migrated within the scheduled change window, but large enough to maximize productivity*. Scheduling too many servers in the same change window means that some servers will not get migrated, thus requiring a change to the global migration plan to accommodate the left-over servers. Scheduling too few servers in the batch leads to underutilization of the migration engineers' time, driving up the cost of migration.
- *Servers that depend on each other should be migrated in the same batch.* Otherwise, the servers would be separated by a slow network link (e.g., one of them is migrated to the target environment, whereas the other stays in the source environment). This may lead to significantly reduced performance.

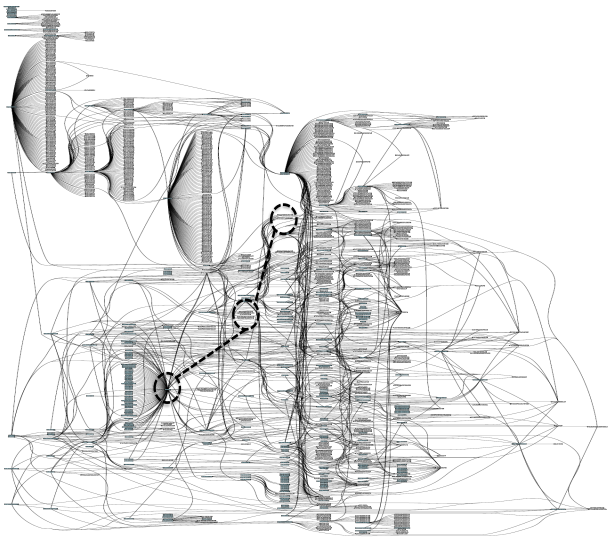

Fig. 2. A portion of the server-to-server dependency graph of a medium scale data center. Dashed circles and lines indicate an example of a three-tier server dependency architecture, from WebShpere Application Server (WAS) → DB2 database server → external storage server.

Traditionally, the task of creating the migration plan is done manually. It requires reasoning about tens of thousands of servers with millions of connections between them. It is, thus, not uncommon for a plan to take weeks, if not months to complete.

Figure 2 shows an example of what a typical connectivity graph looks like for an enterprise IT environment. Dotted circles and lines in Figure 2 indicate a critical three-tier architecture (WebSphere Application Server (WAS) → DB2 → Storage) that is hidden in a messy graph. Thus, the major obstacle in automating this step is identifying critical

dependencies among servers, which is the focus of this paper. By automatically identifying the dependencies between servers and their components, we can significantly reduce the time it takes to devise a migration plan, leading to much reduced migration cost.

III. Model

A. Problem Formulation

We approach the problem of finding relationships among hosts by focusing on finding relationships among the middleware components installed on the hosts. We are interested in finding correlations between occurrences of different middleware components. The problem of discovering correlations is analogous to finding collocations in texts and can be cast as modeling the conditional probability of an event given another event. More formally, we let

$$\forall x, y \in \{MWconfigurations\}, Find \;\; p(x|y),$$

where x and y are members of a set of *middleware* (MW) configurations. Examples of MW are application server, database, etc. The task is to find the conditional distribution of $p(x|y)$. The problem can also be examined from an information retrieval perspective. Suppose one wants to find all the hosts on which a particular middleware y is present. In the hosts that are retrieved, we ask what is the chance that the hosts also contain another middleware x.

The following subsections detail two methods for modeling $P(x|y)$. The first is *mutual information* and the second is a *generalized Kullback-Leibler divergence* method. Additionally, we also present results using a straightforward Bayes formula for conditional probability:

$$p(x|y) = \frac{p(xy)}{p(y)}.$$

B. Mutual Information

Mutual information (**MI**) is a measure of the information overlap between two random variables. In this subsection, we briefly review definitions and properties of MI, and then introduce one of the mutual information methods, pointwise mutual information (**PMI**).

The mutual information between two random variables X and Y, whose values have marginal probabilities $p(x)$ and $p(y)$, and joint probabilities $p(x, y)$, is defined as:

$$I(X;Y) = \sum_{x,y} p(x,y) \ln \frac{p(x,y)}{p(x)p(y)}. \tag{1}$$

The information overlap between X and Y is 0 when the two variables are independent, as $p(x)p(y) = p(x, y)$.

Pointwise mutual information is a measure of how much the actual probability of a particular co-occurence of events $p(x, y)$ differs from the probabilities of the individual events if they are assumed to be independent. Throughout this paper, we use the pointwise mutual information (**PMI**) of a pair $< x, y >$, where $x \in X$ and $y \in Y$:

$$I(x,y) = \ln \frac{p(x,y)}{p(x)p(y)}. \tag{2}$$

C. The Kullback-Leibler Divergence

Kullback-Leibler model is a measure of information on two probability distributions associated with the same experiment. The Kullback-Leibler divergence (*KLD*) measures the difference between two probability distributions over the same problem space. The Kullback-Leibler divergence of the probability distributions p, q on a finite set X is defined as follows:

$$D_{KL}(p||q) = \sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)}. \tag{3}$$

Our idea here is similar to the exponential models constructed by the maximum entropy framework [7]. An exponential model has three components: a prior distribution, a set of questions, and the weights associated with the questions. A prior distribution, $p_0(x|y)$, captures any prior knowledge the modeler may have on the domain. The simplest kind of prior is a uniform distribution signaling there is no prior knowledge. The questions are asked on the joint events, (x, y). The questions are usually binary-valued, although they do not have to be. An example of a binary-valued question is if middleware x and middleware y are both present on the same host. The questions are formally called feature functions. The weights on the questions determine how important or how valuable the questions are in maximizing an objective function. An exponential model combines the three components as follows:

$$p(x|y) = p_0(x|y) \frac{1}{Z(y)} \exp(\sum_i \lambda_i \phi_i(x,y)), \tag{4}$$

where $\{\phi_1, \phi_2, ...\}$ is the set of binary-valued *features* defined on the x and y, and $\{\phi_1, \phi_2, ...\}$ is the set of *weights* associated with the features. There is a λ for each ϕ. Z is a normalization factor, which is computed by summing over all the events in the data and over all the feature functions as in the following equation:

$$Z(y) = \sum_x \exp \sum_i \lambda_i \phi_i(x,y). \tag{5}$$

In the maximum entropy framework, the model $p(x|y)$ must satisfy expectation constraints on the features. The constraint on the model is that the model's expectation of *feature* ϕ_i must be the same as the empirical expectation in the data. Let the model's expectation of ϕ_i be $E\phi_i$, and the expected value of ϕ_i in the data be $\tilde{E}\phi_i$. The constraint is given by the following equation:

$$E\phi_i = \tilde{E}\phi_i, \forall i. \tag{6}$$

The model's expectation $E\phi_i$ and the empirical expectation $\tilde{E}\phi_i$ are given respectively in the following equations in which $\tilde{p}$ denotes the empirical distribution in the data:

$$E\phi_i = \sum_{x \in X, y \in Y} p(x|y)\tilde{p}(y)\phi_i(x,y), \tag{7}$$

and

$$\tilde{E}\phi_i = \sum_{x \in X, y \in Y} \tilde{p}(x,y)\phi_i(x,y). \tag{8}$$

A typical objective function is the likelihood of the data. Log likelihood of the data is given in the following equation:

$$\sum_i = \ln p(x_i|y_i). \tag{9}$$

Hosts	MW_CLASS	MW_SUBCLASS	VENDOR	SERVICE_PACK	DISTRIBUTION_NAME	INSTALL_PATH
cam-db-x	DBMS	DB2	IBM	8.2	DB2 Enterprise Edition	../opt/db2
cam-db-x	FS	JFS2	IBM			
cam-db-x	AIX	AIX_HACMP	IBM		AIX HACMP cluster manager	../sbin/cluster
...	...	...	...	...	...	...
cam-de-y	APS	WAS	IBM	6.0.2.17	IBM WebSphere APP Server	../WAS/AppServer
cam-de-y	WEB	IHS	IBM	6.0.2.15	IBM HTTP Server	../usr/IBMIHS
...	...	...	...	...	...	...

TABLE I. EXAMPLE OF THE HOST MIDDLEWARE INSTALLATION DOCUMENT. COLUMN MW_CLASS IS A GENERAL, BROAD, AND VENDOR-INDEPENDENT CLASS. FOR EXAMPLE, DBMS STANDS FOR DATABASE MANAGEMENT SYSTEM, AND APS STANDS FOR APPLICATION SERVER. COLUMN $MW\_SUBCLASS$ HAS VENDOR-DEPENDENT MIDDLEWARE CORRESPONDING TO THE MW_CLASS. FOR EXAMPLE, DBMS_DB2, OR APS_WAS CAN DESCRIBE A MIDDLEWARE COMPONENT.

Maximizing the data likelihood is equivalent to minimizing the Kullback-Leibler divergence between the prior P_0 and the conditional model P that satisfies the expectation constraints. In many applications (such as [8] and [9]), the normalization factor Z is unnecessary. However, one cannot simply drop Z because to maximize the data likelihood, one simply needs to increase the λs and the likelihood can be increased indefinitely. To properly remove the normalization factor Z, a new well-defined optimization function is needed. Several researches have reported extensions to the entropy maximization and generalized Kullback-Leibler divergence. Previous uses of these extended models include a variety of topics including logistic regression, boosting and information retrieval. If we define an unnormalized non-negative exponential function T ([8]):

$$T = \{T(x|y) = T_0(x|y)e^{\lambda(x,y)}\}, \tag{10}$$

and a generalized Kullback-Leibler divergence [10]:

$$D(T_1||T_2) = N - T(1) + \sum_{i=1}\sum_{x\in X} T_1(x|y_i)\ln\frac{T_1(x|y_i)}{T_2(x|y_i)}, \tag{11}$$

where N is the number of events in the data and $T(1)$ denotes a constant function, then the optimal weights of the scalar features which are subject to the expectation constrains have a closed form solution as follows:

$$\lambda^* = \ln\frac{E[\phi]}{\tilde{E}[\phi]}, \tag{12}$$

where $E[\phi]$ and $\tilde{E}[\phi]$ are the model's empirical expectations as defined in Equations (7) and (8), respectively. The optimal gain over the prior is thus as follows:

$$G = L(\lambda^*) - L(0). \tag{13}$$

D. Compute Features and Expectations

We now use a concrete example to show how to compute the features and expectations. Suppose $\mathcal{S} = \{MW_1 \ldots MW_N\}$ is the collection of all occurrences of all middelware components. Table I shows an example of a set of hosts $\mathcal{H}$ and their installed middlewares MW_N. Further suppose that we have no prior knowledge and that any member of $\mathcal{S}$ is equally likely to co-occur with any given middleware MW_i. That is, $p(MW_j|MW_i) = \frac{1}{N}$. We define a binary feature function:

$$\phi(MW_i, MW_j) = \begin{cases} 1 & \text{if } MW_i \text{ and } MW_j \text{ on the same } H_i \\ 0 & otherwise. \end{cases} \tag{14}$$

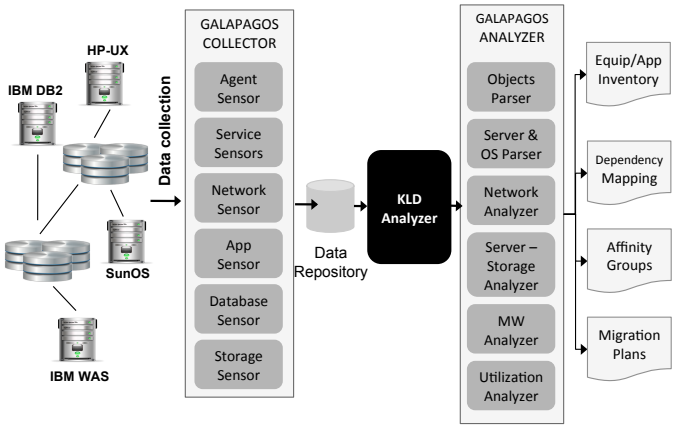

Fig. 3. Galapagos Discovery Framework

Note that this feature is separable as follows:

$$\phi(MW_i, MW_j) = \phi_1(MW_i)\phi_2(MW_j). \tag{15}$$

Equation (15) is equivalent to Equation (14) because MW_i and MW_j co-occur if and only if they both are found on the same host. Then, the empirical expectation of feature (14) is thus:

$$d_0 = \sum_{i=1}^{N}\phi(MW_i, MW_j) = N_{ij}, \tag{16}$$

where N_{ij} is the number of hosts containing both MW_i and MW_j. The model's expectation of feature (14) is then:

$$d = \sum_{i=1}^{N}\sum_{MW\in\mathcal{S}}\frac{1}{N}\phi_1(MW_i)\phi_2(MW_j). \tag{17}$$

Once d_0 and d are thus computed, we proceed to compute λ^* and the optimal gain according to Equations (12) and (13).

IV. EVALUATION

A. Implementation

Our implementation builds on Galapagos [2], an existing discovery framework, as shown in Figure 3. Galapagos includes a light-weight discovery front-end that collects information from servers and a back-end that processes and analyzes the collected data to provide high-level insights into the IT environment.

We implemented—using a mixture of Perl and C—the three algorithms outlined in Section III: mutual information, Kullback-Leibler Divergence, and naive Bayesian. Using the

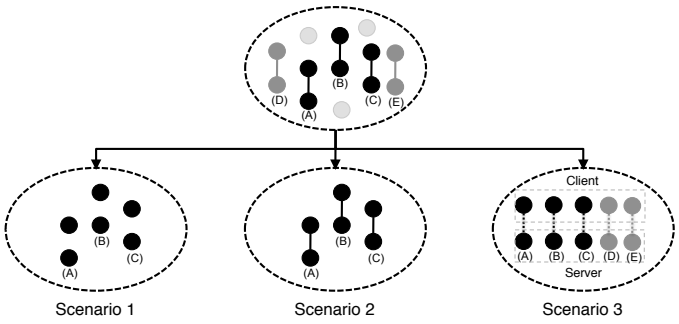

Fig. 4. Three Scenarios: *Hodgepodge*, *Sausage*, and *Client-Server* Relations.

data collected from Galapagos, we then applied these algorithms to compute the dependencies of middleware components among hosts.

B. Application Scenarios

In addition to computing the middleware correlation among all hosts, we also investigate how the correlations change in different partitions of the hosts. Some hosts connect to (or depend on) other hosts for certain middleware, application, or service. From this dependency information, we can divide the space of all hosts into a variety of subspaces and examine the correlation of middleware components in each subspace.

We describe two different subspaces based on the dependencies among the hosts. Note that a middleware MW can be described as its $MW\_CLASS\_MW\_SUBCLASS$ as shown in Table I. For readability, we only use $MW\_SUBCLASS$ in the rest of this paper (e.g., 'WAS', 'DB2', and 'MSQ', etc.).

Scenario 1: Hodgepodge Relation. Suppose there are k dependency types among all the hosts. In the first scenario, we group together those hosts engaged in the same dependency $\mathcal{D}$. We call this smaller group of hosts H_D. We then compute correlations within the group. The correlations we compute are still within each host and do not depend on $\mathcal{D}$. The dependency type $\mathcal{D}$ is only used to define the group and is not retained in further computation. Each host in H_D is treated equally and separately, hence the name *hodgepodge*. In essence, this scenario computes intra-host correlation on a smaller space.

Scenario 2: Sausage Relation. Unlike Scenario 1, this scenario retains the dependency information $\mathcal{D}$ and the correlations are computed across hosts. The question we ask here is what is $P(MW_i|MW_j)$, where MW_i is on $host_i$ and MW_j is on $host_j$. Furthermore, $host_i$ and $host_j$ have the dependency $\mathcal{D}$. This scenario computes inter-host correlations within the same dependency type $\mathcal{D}$. Figure 4 is a pictorial illustration of this scenario. The links in the figure resemble sausages, hence the name *sausage*.

Scenario 3: Client-Server Relation. A third way to group the hosts is to group all the clients together and all the servers together. Each dependency type $\mathcal{D}$ involves two hosts which can be conceptually thought of as a client and a server. This scenario 3 is called *client-server*. Correlations are then computed within the client and server subgroups, respectively.

Accounts	A	B	C1	C2	D
Servers	100	300	500	1000	500
OS Types	3	6	8	9	6
DB Types	3	4	4	5	3
HW Models	10	15	38	45	35

TABLE II. HIGH LEVEL CHARACTERISTICS OF DATASETS

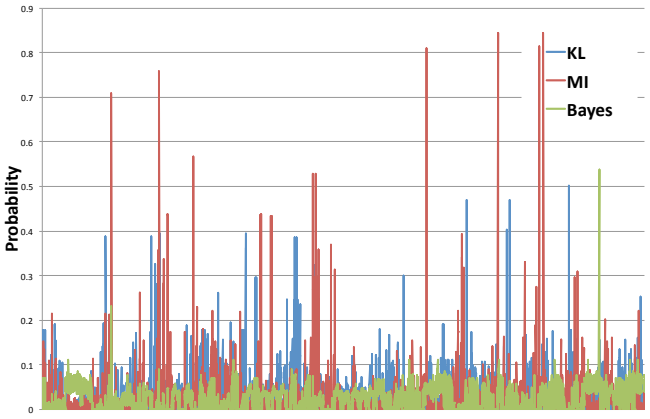

Fig. 5. Comparison between Bayes, PMI, and KLD

C. Experimental Setup

We choose five datasets from 20 large migration projects. Table II summarizes the high level description of the corresponding data centers. We can see that each data center is a very complex environment that runs on multiple operating systems, hardware, and utilizes different databases for mission-critical applications. Each server among the chosen data centers also contains multiple installations of different middleware components, such as various management tools, Web services, and storage systems, etc.

D. Experimental Results

Comparison of Three Conditional Probability Methods. These are Bayes, Pairwise Mutual Information (PMI), and Kullback-Leibler Divergence (KLD). We choose one account data from Country C1, which contains 500 hosts and over 4K middleware pairs. Figure 5 plots the distributions of the three methods across all middleware correlations. From the figure, we can see that Bayes is flat except for a few spikes at the beginning and the end. PMI has more variations: too spiky and more fluctuating. KLD has more discerning power in terms of more spikes, and is more consistent from the beginning to the end. The same phenomenon is observed across other accounts. In the remainder of this section, we demonstrate that how KLD method can help the migration practitioner to execute migration tasks.

Geographic Differences. In this experiment, we choose two pairs of accounts from different national geographic countries, Country A *vs.* Country B (Figure 6), and Country C1 *vs.* Country C2 (Figure 7). As shown in Figure 6, we see that the distribution of the paired middleware components are significantly different (e.g., $P(A|B)$ given B = MSQ) between these two accounts. In this case, only 2 out of 10 pairs of middleware components are similar (*Windows*, *VMTOOLS*). From Figure 7, we can see that the distribution of paired middleware components between two accounts from the same country are very similar. In this case, 7 out of 9

Country A, P (* | MSQ)	
MiddleWare (*)	Probability
Windows	**0.24**
IIS	0.17
IBM-DIRECTOR	0.093
DISK	0.087
SYS	0.076
NET	0.076
JP1	0.064
MSEX	0.058
VMTOOLS	**0.038**
MSCCM	0.025

Country B, P (* | MSQ)	
MiddleWare (*)	Probability
ClearCase	0.19
CIFS	0.18
DB2	0.087
Windows	**0.062**
SMB	0.059
UNKNOWN	0.059
NTFS	0.059
SYM_SF	0.057
AVSYM	0.057
VMTOOLS	**0.050**

Fig. 6. Middleware configuration differences of two clients in different geographic areas, given $P(*|MSQ)$. Country A vs. Country B

Country C1, P (* | MSQ)	
MiddleWare (*)	Probability
NTFS	0.094
Windows	0.094
MSEX	0.092
AVSYM	0.085
SYM_SF	0.076
CSRSS	0.065
HPSMH	***0.059***
IIS	0.058
HPSYS	***0.057***

Country C2, P (* | MSQ)	
MiddleWare (*)	Probability
NTFS	0.093
Windows	0.092
CSRSS	0.092
AVSYM	0.085
MSEX	0.080
MS-WMI	***0.068***
MSCCM	***0.059***
SYM-SF	0.054
IIS	0.052

Fig. 7. Middleware configuration differences of two clients in the same geographic areas, given $P(*|MSQ)$. Country C1 vs. Country C2

pairs of middleware components (those different middleware components are in bold font as shown in Figure 7). The same observation is found across all other middleware pairs (given different B) and all our clients globally. Reasons that cause the similar middleware configurations in the same geographic areas may due to the common practice and regulations applied in that area, and require further investigation. However, this information is already very useful to the migration teams since usually one migration team will simultaneously work on multiple accounts which are geographically different, and proper preparation and practice can be taken accordingly.

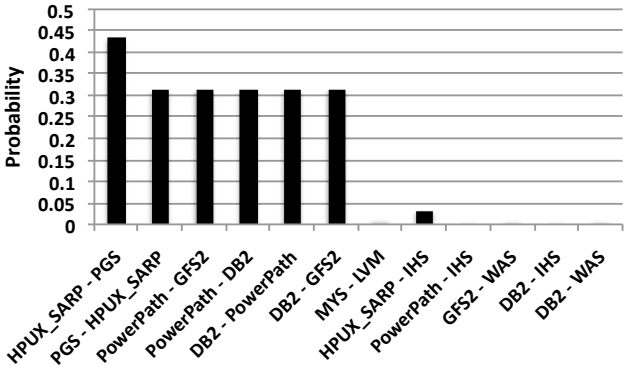

Fig. 8. Scenario 1: Hodagepodge relation in IHS-WAS subspace. X-Y axes show P(A|B). For example, given B= $Powerpath$, P(DB2 | B) = 31%.

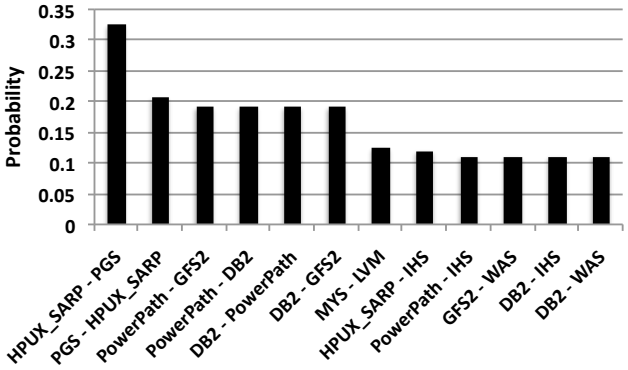

Fig. 9. Scenario 2: Sausage relation in IHS-WAS subspace. X-Y axes show P(A|B). For example, given B= WAS, P(DB2 | B) = 12%.

E. Results of Three Scenarios with KBL

Scenario 1. The Hodgepodge relation represents the intra-host middleware correlation. In Figure 8, we show the distribution of paired middleware components within the *IHS-WAS* group. We can see that some paired middleware component clearly stand out from others. For example, given we have seen the DB2 database, we have over 30% probability that we also see EMC *Powerpath* server-resident management tool installed on the same host to enhances performance and information availability, in this case to enhance the performance of DB2. We can also see that, given we see IBM *WAS*, we have a slim chance, around 1% probability, to see DB2 database co-occur on the same host. This finding confirms the common practice in a normal IT environment that IBM WebSphere and DB2 database are not installed on the same host. However, if it does happen, like in this case, the practitioner needs to pay attention. Similar findings are observed across all other different subspaces and accounts. Hence, this intra-host correlation analysis is very useful to the migration team.

Scenario 2. The Sausage relation represents the inter-host middleware correlation. In Figure 9, we show the distribution of paired middleware components within the same *IHS-WAS* group, but across different hosts. We can see that, for example, given that we have seen the DB2 database, we have around 18% probability to see EMC *Powerpath* toolkit, but on different host. In comparison to intra-host analysis (Scenario 1), the probability number decreases and tell us that this pair middleware components can either co-exist on the same host, or among different hosts (e.g., via network storage system). In addition, we see that, given we see IBM *WAS*, we have a high chance, around 10% probability, to see DB2 database occur on the another host. Compared with the same middleware pair, P(DB2 | WAS), in Hodgepodge scenario, the probability in sausage scenario increase 10 times. Similar phenomenon is also observed in different middleware subspace as shown in Figure 10 and Figure 11. For example, given that we see WAS, we have 20% probability to see logical volume manager (LVM) on a different host in the same middleware group as shown in Figure 10.

Figure 12 demonstrates the comparison results between Hodgepodge and Sausage relation. As expected from multi-tier applications, in which critical middleware components are

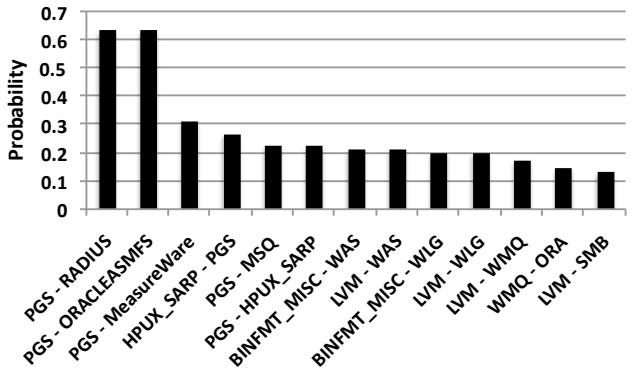

Fig. 10. Scenario 2: Sausage relation in Net-WAS subspace. X-Y axes show P(A|B). For example, given B= WAS, P(LVM | B) = 21%.

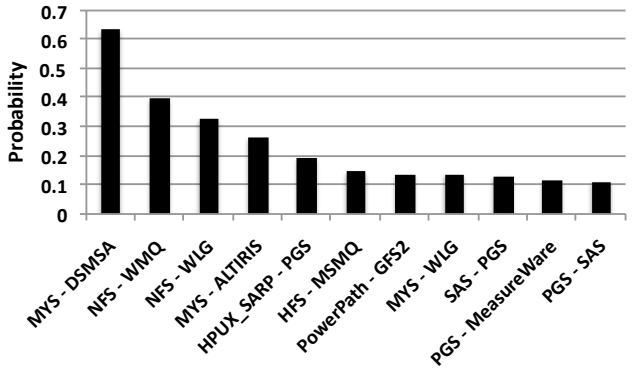

Fig. 11. Scenario 2: Sausage relation in Net-ORA subspace. X-Y axes show P(A|B). For example, given B= WLG (WebLogic), P(NFS | B) = 33%.

installed at different hosts, the association distribution can be very different. The grey bars represent Hodgepodge relation and black bars are for Sausage, and the notation DB2-WAS illustrates the conditional probability of DB2 given by WAS. In this last bar chart of the figure, the DB2-WAS, it clearly demonstrates that P(DB2|WAS) is much higher in Sausage relation than in Hodgepodge. It means that DB2 and WAS are more likely installed at two different servers than installed at a single server. This aligns with common practices. Interestingly, notice that *P(DB2|WAS)* and *P(Powerpath|DB2)* indicate a critical three tier architecture. The latter 6 middleware pairs in the chart, from MYS-LVM to DB2-WAS, have the same behavior. The first 6 middleware pairs (HPUX_SARP - PGS to DB2-GFS2) are slightly likely to be installed at the same server. By checking the nature of these software pairs, it is not critical to install them at different servers. For different data center operations, they can go either way. These inside finding will be very helpful for migration planning.

Scenario 3. The Client-Server relation investigates the similarities and differences between client and server configurations. Correlations are computed for the clients and servers in each dependency group $\mathcal{D}$. For each group, the distributions of the clients and servers are compared. In addition, we also add the overall middleware distribution of all hosts into the comparison mix. The overall distribution is the one computed is Section III-C, where all hosts from the same account are put into one group. The overall distribution contains more middleware

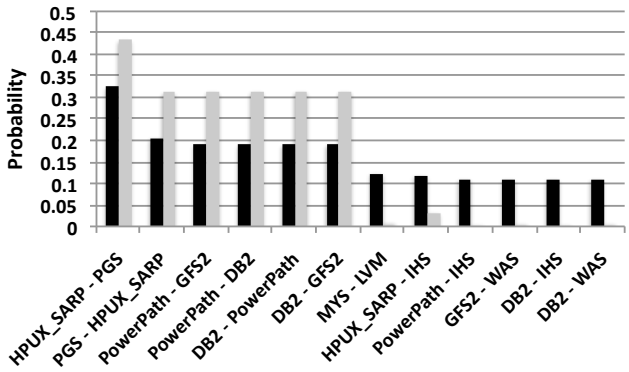

Fig. 12. Comparison of Hodgepodge & Sausage relation in IHS-WAS subspace. X-Y axes show P(A|B). For example, given B= WAS, P($DB2$ | B) = 12%. Grey bar is the Hodgepodge relation. Black Bar is the Sausage relation.

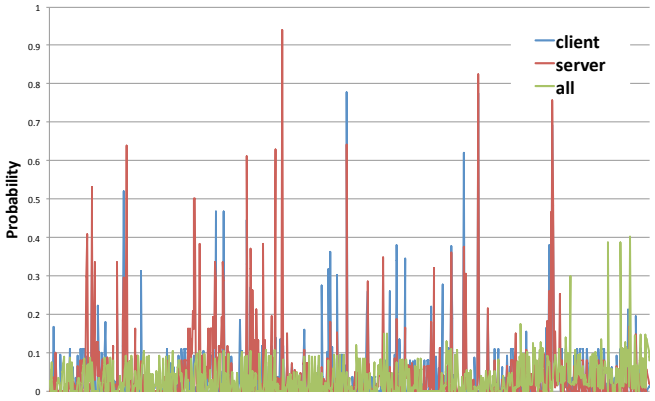

Fig. 13. Client-Server NET-SMB relation

pairs than either the client or the server distribution in each dependency group $\mathcal{D}$. Only the middleware pairs present in the client, server, and overall distributions are extracted for comparison. Figure 13 plots the three distributions based on NET-SMB dependency. Figure 13 shows that the client and server distributions are not far away from each other, but both are very different from the overall distribution.

V. Related Work

A. Cloud Computing

Cloud computing is not just about a technological improvement in data centers, it represents a fundamental change in how IT is provisioned and used [11]. For enterprises to use cloud computing, they have to consider the benefits, risks and effects of cloud computing on their organizations. Armbrust et al. [12] argued that elasticity is an important economic benefit of cloud computing as it transfers the costs of resource over-provisioning and the risks of under-provisioning to cloud providers. Motahari Nezhad et al. [13] added that the potentially reduced operational and maintenance costs is also important from a business perspective. However, little has been published about the organizational risks of the change that cloud computing brings to enterprise, nor has been a systematic study of the complexity of IT transformation to cloud environments.

B. IT Infrastructure Discovery

Today IT asset and dependency discovery tools are available from many vendors. They probe network nodes with requests [4], monitor network traffic [3], or analyze software configurations [2], [5]. The configuration analysis is done for packaged middleware and applications such as databases, Java Enterprise Edition (Java EE) servers, and Enterprise Resource Management (ERM) systems. For Java EE servers, only the objects and relations explicitly configured at the server level are analyzed, such as which Enterprise JavaBeans (EJBs) are deployed and what resources are declared.

Other research try to observe dependencies at the operating-system level. On Solaris systems, this is in fact an option via DTrace [14]. Unfortunately, on other operating systems no such mechanism is commonly available. The *lsof* utility [15] comes closest, but it is not typically installed. There has been research on instrumenting other operating system kernels to understand storage dependencies [16], [17]. However, these tools are typically not available in the enterprise optimization scenarios that we investigate, and installing an operating-system level tool on production servers would be an equally large project as the optimization scenarios themselves, and again require detailed prior discovery and analysis of potential impact. Provenance-aware file systems such as PASS [18] imply an even larger operating-system change, because they modify the process-to-filesystem interaction to obtain comprehensive records of past dependencies between these two layers. While they are very interesting, they are certainly not found on the enterprise servers that we are optimizing.

Existing research [19], [20] focus on methods to discover cross-domain relationships in distributed systems, either by statistically analyzing system behavior [19], on the basis of observation of system activity, or by using system support (e.g., passing tokens or other metadata over communication between layers [20]). In addition, several commercial tools focus on discovery of infrastructure assets by scanning a range of IP addresses and querying the systems that respond [6], [5]. Network communication relationships among applications are discoverable by capturing network packets and analyzing their headers [21]. Various systems have investigated building distributed system-dependency graphs using passive methods such as trace collection and offline analysis [19], [20]. Some of the uses of a dependency graph include problem determination, performance analysis, and visualization. Our system and approach differ from these approaches in that it specifically discovers a finer-grain scope of dependency between software components, application-to-server, and server-to-server.

VI. Conclusion

In this paper, we introduce a novel Kullback-Leibler (KL) divergence based method that can systematically discover the complex server-to-server, application-to-server relationships. We evaluated the methods using five real client datasets (from large enterprise migration efforts). We demonstrate how a generalized KL in the unnormalized maximum entropy framework performs better than standard techniques such as mutual information. We also show results on mined middleware correlations. Furthermore, analyses on subsampling hosts based on their dependencies uncover intriguing phenomena in different subspaces. These analyses can aid migration engineers in a variety of tasks ranging from migration planning to investigating failures, and significantly reduce migration cost.

References

[1] C. F. Workgroup, "Configuration management database (cmdb) federation specification." *http://xml.coverpages.org/DMTF-DSP0252-CMDB-Federation.pdf.*, 2009.

[2] K. Magoutis, M. Devarakonda, N. Joukov, and N. G. Vogl, "Galapagos: model-driven discovery of end-to-end application-storage relationships in distributed systems," *IBM J. Res. Dev.*, vol. 52, no. 4, pp. 367–377, Jul. 2008.

[3] A. Caracas, A. Kind, D. Gantenbein, S. Fussenegger, and D. Dechouniotis, "Mining semantic relations using netflow," in *BDIM'08*, 2008, pp. 110–111.

[4] ISI-Snapshot, "Agent-less accurate and rapid it infrastructure inventory, configuration and utilization collection using a single tool." *www.isiisi.com*, 2011.

[5] HP-DDMA, "Hp discovery and dependency mapping advanced (ddma)." *http://www8.hp.com/us/en/software-solutions/software.html?compURI=1175751.*, 2010.

[6] IBM-Tivoli-TADDM, "Tivoli application dependency discovery manager." *http://www-306.ibm.com/software/tivoli/products/ taddm.*, 2008.

[7] S. Della Pietra, V. Della Pietra, and J. Lafferty, "Inducing features of random fields," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 4, pp. 380–393, Apr. 1997.

[8] K. Papineni, "Why idf," in *NAACL*, 2001, pp. 516–519.

[9] S. F. Chen, K. Seymore, and R. Rosenfeld, "Topic adaptation for language modeling using unnormalized exponential models," 1998.

[10] S. Dharanipragada, M. Franz, J. S. McCarley, K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Statistical methods for topic segmentation," in *INTERSPEECH*, 2000, pp. 516–519.

[11] M. Creeger, "Cto roundtable: Cloud computing," *Communications of the ACM*, vol. 52, no. 5, pp. 56–60, Jun. 2009.

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," UC Berkeley, CA, Tech. Rep., 2009.

[13] H. R. Motahari-Nezhad, B. Stephenson, and S. Singhal, "Outsourcing business to cloud computing services: Opportunities and challenges." *HP Lab.*, 2009.

[14] "Dtrace," in *http://hub.opensolaris.org/bin/view/Community+Group+dtrace.*, ser. Sun Microsystems, 2005.

[15] V. A. Abell, "Lsof - list open files, unix-like system command line." *http://people.freebsd.org/ abe/.*, 2012.

[16] A. Aranya, C. P. Wright, and E. Zadok, "Tracefs: A file system to trace them all," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, ser. FAST '04, 2004, pp. 129–145.

[17] N. Joukov, A. Traeger, R. Iyer, C. P. Wright, and E. Zadok, "Operating system profiling via latency analysis," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI '06, 2006, pp. 89–102.

[18] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ser. ATEC '06, 2006.

[19] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ser. SOSP '03, 2003, pp. 74–89.

[20] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services," in *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, ser. DSN '02, 2002, pp. 595–604.

[21] A. Kind, D. Gantenbein, and H. Etoh, "Relationship discovery with netflow to enable business-driven it management," in *BDIM'06*, 2006, pp. 63–70.