Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud

Mohammad Hajjat†, Xin Sun†, Yu-Wei Eric Sung†, David Maltz‡, Sanjay Rao†
Kunwadee Sripanidkulchai⋆, and Mohit Tawarmalani†
†Purdue University, ‡Microsoft Research, ⋆IBM T.J. Watson Research Center

ABSTRACT

In this paper, we tackle challenges in migrating enterprise services into hybrid cloud-based deployments, where enterprise operations are partly hosted on-premise and partly in the cloud. Such hybrid architectures enable enterprises to benefit from cloud-based architectures, while honoring application performance requirements, and privacy restrictions on what services may be migrated to the cloud. We make several contributions. First, we highlight the complexity inherent in enterprise applications today in terms of their multi-tiered nature, large number of application components, and interdependencies. Second, we have developed a model to explore the benefits of a hybrid migration approach. Our model takes into account enterprise-specific constraints, cost savings, and increased transaction delays and wide-area communication costs that may result from the migration. Evaluations based on real enterprise applications and Azure-based cloud deployments show the benefits of a hybrid migration approach, and the importance of planning which components to migrate. Third, we shed insight on security policies associated with enterprise applications in data centers. We articulate the importance of ensuring assurable reconfiguration of security policies as enterprise applications are migrated to the cloud. We present algorithms to achieve this goal, and demonstrate their efficacy on realistic migration scenarios.

Categories and Subject Descriptors: C.4 [**Performance of systems**]: Modeling techniques ; C.2.3 [**Computer communication networks**]: Network operations– Network management

General Terms: Algorithms, Design, Experimentation, Management, Performance, Security

Keywords: Cloud Computing, Enterprise Applications, Security Policies, Performance Modeling, Network Configurations

1 Introduction

Cloud computing promises to reduce the cost of IT organizations through lower capital and operational expense stemming from the cloud's economies of scale, and by allowing organizations to purchase just as much compute and storage resources as needed, only when needed. Many cloud early adopters have had great success in leveraging the cloud to deliver new services as an alternative to implementing their own server infrastructures [4, 17].

The advantages and initial success stories of cloud computing are prompting many enterprise networks to explore how the cloud could be leveraged to deliver their existing enterprise applications. Consider a recent survey of 1,780 data center managers in 26 countries worldwide conducted by Symantec [25]. Over 36% of respondents indicated that the large number of applications and complexity of managing data centers were huge problems that they faced. Over 82% of respondents indicated that reducing data center costs was one of the most important objectives for coming years. Over 72% of respondents indicated they were considering or using public cloud computing, although 94% of these respondents were still in the discussion, planning, trial or implementation stages.

Despite the significant interest, migrating enterprise applications to cloud computing is a major challenge (e.g., [11]). On the one hand, enterprise applications are often faced with stringent requirements in terms of performance, delay, and service uptime. On the other hand, little is known about the performance of applications in the cloud, the response time variation induced by network latency, and the scale of applications suited for deployment. Further, industry-specific regulations (e.g., in health care industries), and national privacy laws may restrict what data an enterprise may migrate to the cloud [11].

In response to these concerns, there has been significant interest in the industry in hybrid architectures where enterprise applications are partly hosted on-premise, and partly in the cloud (e.g., [15,22]). Enterprise applications are typically composed of multiple components, and hybrid architectures allow for individual components to be migrated, or kept local. Hybrid architectures offer enterprises flexibility in decision making that can enable them to find the right balance between privacy considerations, performance and cost savings. For instance, sensitive databases (e.g., related to credit card processing) may be located local to the enterprise, while relatively less sensitive components could be migrated to the cloud. Users external to the enterprise could be handled through servers deployed in the cloud, while internal users could be handled through servers located on premise.

In this paper, we take a first step towards articulating and addressing two challenges involved in enabling enterprises to move to such a hybrid cloud model, as we discuss below:

Component placement: While hybrid architectures offer several advantages, deciding which components must be kept local, and which components must be migrated is non-trivial. The challenge stems from the fact that enterprise applications consist of a large number of components, with complex interactions, as our examination of applications deployed in a global Fortune 100 company, and a large university indicate. To address this challenge, we present

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.
Copyright 2010 ACM 978-1-4503-0201-2/10/08 ...$10.00.

a model that enables application architects to systematically plan which components of their application must be migrated. Our model takes into account enterprise policies, cost savings from migration, and increased transaction delays and wide-area communication that may result from the migration. Our model also considers the data flows between application components, and the spread of users (e.g., if they are located within or outside enterprise premises). We evaluate our algorithms using (i) a real cloud-based application on an Azure-based cloud test-bed; and (ii) a model of a real enterprise application deployed in a large-scale campus network. Our results show there are situations where hybrid migrations are optimal, and reveal the importance of a planned approach to determining which components to migrate.

ACL Migration: While executing an actual migration entails several challenges [23,28], security related issues are perhaps the most vexing. To highlight the importance, consider that over 83% of data center managers in the survey mentioned above have indicated security is the most important initiative for their data centers in coming years [25]. Our examination of an operational data-center of a large campus network indicates that access to servers corresponding to enterprise applications is highly restricted, and the security (reachability) policies are often encoded in low-level device configurations. A key barrier to realizing hybrid migrations is the need to ensure that the reachability policies continue to be met. We present automated approaches to reconfigure network devices in an assurable fashion. Our approach involves extracting the end-to-end policies from low-level device configurations, transforming them (if necessary) to handle changes to address space assignments, and placing the transformed policies in the migrated setting. We evaluate our algorithms using realistic migration scenarios, and router configurations of a large campus network. Overall, our results show the feasibility and importance of our approach.

While our results are promising, our work is but a first step towards addressing challenges in migrating enterprise networks to the cloud. We discuss other challenges not considered in the paper, and open issues in § 7.

2 Motivation

In this section, we begin by exposing the complexity of real enterprise applications today. This leads us to formulate important challenges in migrating enterprises to cloud-based models.

2.1 Characteristics of enterprise applications

Enterprises run a number of applications to support their day-to-day business. For instance, a company may run applications to support payroll, travel and expense reimbursement, customer relationship management, and supply chain management. Similarly, a university may run applications corresponding to student registration, employee payroll, class management, and alumni relations. Such applications are implemented using multi-tier architectures. Typically, applications can be decomposed into three functional tiers: (i) a front-end tier, containing web servers that handle user requests and application presentation; (ii) a business-logic tier, containing servers that perform specialized application logic such as sales analysis; and (iii) a back-end tier that comprises of various database servers. While a 3-tiered design is the conventional architecture used in most applications, in practice applications are much more complex. Each of these tiers may have multiple functional components; each component may have multiple servers performing the same role and executing the same code while load-balancers are employed to spread the requests across each server.

Fig. 1 depicts the number of distinct functional components for

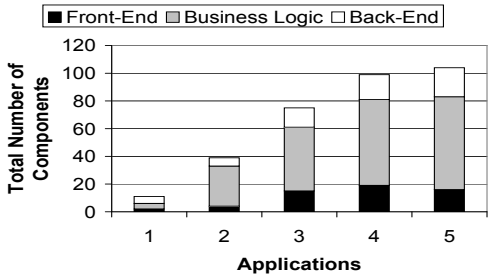

Figure 1: Distinct functional components for five enterprise applications.

five enterprise-wide applications sampled out of hundreds of such applications used by a Fortune 100 company, broken down by tier. Note that we only include distinct functional components – server replicas that run the same code are not counted. The total number of distinct components in each application varies from 11 to over 100 components depending on the nature of the application. In some of these applications, there are up to 19 distinct front-end components, 67 business-logic components, and 21 back-end datastores. Furthermore, the flow of requests between these components is often complex. Depending on the specific type of user transaction, the flows could touch upon different functional components across all the tiers. To further appreciate the complexity, we refer the reader to § 5.2 (Fig. 10) for a detailed model of an enterprise application deployed in a large campus network.

An enterprise application could be accessed by two types of users: (i) users internal to the enterprise; and (ii) users external to the enterprise. For instance, in a university campus setting, current employees and students correspond to internal users, but alumni, and prospective students correspond to external users.

2.2 Security policies in enterprise data centers

Many enterprise applications (e.g., employee payroll) are extremely sensitive, and access to servers corresponding to these applications is highly restricted. For instance, following the tiered application architecture, typical security policies may only allow front-end servers to access business logic and back-end servers. The security policies are carefully crafted reflecting the complex application interdependencies so only application components that need to talk to each other are permitted to do so. Servers corresponding to various enterprise applications are logically partitioned into Virtual LANs (VLANs). Each VLAN is protected by a *firewall context*. Each firewall context is associated with one or more access control lists (ACLs). An ACL is a sequential collection of permit and deny conditions, called ACL rules, and flows to be permitted/denied are identified in terms of their source/destination IP addresses and ports, and protocol, i.e., a 5-tuple (sip,dip,$sport$,$dport$,$proto$). The reader is referred to §5.3 which presents high-level characteristics of the security policies of a data center in a large university campus. The data center has over 40 firewall contexts, each context associated with a pair of ACLs, and with each ACL having several tens and sometimes hundreds of ACL rules.

2.3 Issues in migrating enterprise applications

In this paper, we focus on hybrid cloud architectures, where individual application components may be placed locally or in the cloud. Further, we allow for placement strategies where only a subset of servers in a component are placed remotely. Fig. 2 depicts an example, where only the back-end, and a subset of front-end servers have been moved to the cloud.

Multiple factors can motivate such hybrid deployments. From a performance perspective, migrating the entire application to the cloud is likely to result in higher response times to users inter-

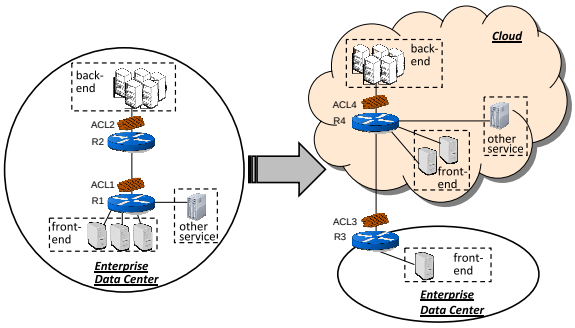

Figure 2: An example hybrid cloud deployment. The figure also illustrates the need to reconfigure security policies.

nal to the enterprise, as well as extensive wide-area communication. Replicating servers both locally and remotely could allow for the two classes of users (internal and external) to be served from different locations. From a data privacy perspective, enterprises may wish to store sensitive databases (e.g., a database that stores credit card information) locally. This may in turn make it desirable to place other components that extensively interact with such databases local, from the perspective of reducing wide-area communication costs and application response times. Finally, while we do not explore in this paper (but briefly discuss in § 7), hybrid architectures may enable enterprises to better handle peak workloads. In particular, servers may be instantiated in the cloud when needed, while requests are in general served from the local data-center.

In this paper, we address two key challenges associated with realizing hybrid cloud architectures:

Planning which servers to migrate: Planning a hybrid cloud layout is a complex problem, where an application architect must take several factors into account. The key factors are (i) honoring policy constraints regarding which components must be migrated; (ii) ensuring application response times continue to meet desired targets; and (iii) ensuring the cost savings from migration are as high as possible. The cost savings in turn depend both on compute and storage components migrated, as well as the wide-area communication costs involved. The transaction delays as well as wide-area communication costs in turn depend on (i) the location of users, i.e. whether they are local to the enterprise, or external to it; and (ii) the nature of interaction between components, i.e., which components interact, and how much traffic they exchange.

Ensuring correctness of security policies on migration: On migrating application servers to the cloud, operators need to reconfigure ACLs to ensure that the security policies are still met. Consider the migration scenario in Fig. 2 with a security policy which only permits front-end servers to reach back-end servers. Ensuring that this policy continues to be upheld on migration requires operators to change both the location of ACLs, as well as rules in the ACLs themselves. In doing so, the primary consideration is to correctly achieve security policy goals. A second consideration is to filter unauthorized packets as early as possible. For instance, if traffic is not permitted from certain enterprise users to a server located in the cloud, it is desirable to ensure that the unauthorized traffic is filtered at the enterprise edge itself rather than filter it after it has traversed the wide-area link to the cloud. Finally, the problem is further complicated due to reassignment of IP addresses after migration as is the practice of certain cloud providers today.

3 Planning a migration strategy

In this section, we present a model that enables application architects to systematically plan which components of their application must be migrated to the cloud. We present a high-level problem formulation in §3.1 and provide modeling details in the remaining subsections.

3.1 Abstraction and problem formulation

Consider an enterprise running K applications, $A_i, 1 \leq i \leq K$, and m components (across all applications), $\mathrm{C}_i, 1 \leq i \leq m$. Each application involves a subset of components. For instance, $A_1 = \{\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_5\}$ indicates that application A_1 has components C_1, C_2, and C_5. We abstract the application structure using a graph $G = (V, E)$. Let $V = \{\mathrm{C}_i\}_{i=1}^{m} \cup \{I, O\}$, where I and O represent internal and external users respectively. Nodes i and j are connected by an edge (i, j) if they communicate. The number of transactions per second and the average size of transactions along (i, j) are denoted by $T_{i,j}$ and $S_{i,j}$ respectively. $T_{i,j}$ and $S_{i,j}$ are the ij^{th} elements of the *transaction matrix*, $\mathbf{T}$, and the *size matrix*, $\mathbf{S}$, respectively. We assume that $\mathbf{T}$ and $\mathbf{S}$ are available.

In deciding how a component must be migrated to the cloud, it is important to consider the specific nature of that component. Web front-ends and business-logic tiers typically consist of multiple servers performing the same role, and easily allow for solutions where a subset of servers may be migrated to the cloud. Back-end databases on the other hand involve storage, and an associated set of servers. Partial migration of a subset of servers involves either (i) replication of storage in the local and cloud data-centers, and the need for techniques to maintain consistency across replicas; or (ii) maintenance of the database in a distributed fashion, a hard problem. In this paper, we assume back-end databases are migrated in an atomic fashion, i.e., all associated storage and servers are either maintained local, or entirely migrated. In the future, it might be interesting to consider partial migration of databases.

More generally, each component C_i is modeled as consisting of N_i servers. The overall goal of the formulation is to determine the number of servers n_i ($n_i \leq N_i$) which must be migrated to the cloud for each component C_i. A database component C_d is easily captured in this model by setting $N_d = 1$, and imposing integrality requirements on n_d.

Let $\mathbf{P}$ denote a set of policy constraints that govern the migration process. For instance, the constraints may indicate that certain application components (e.g., relating to sensitive data storage) are not to be migrated to the cloud. Formally, the goal is to determine a migration strategy $M = \left(n_i\right)_{i=1}^{m}$, such that:

$$\begin{array}{lll} \text{(MP)} & \max & \mathrm{Benefits}(M) - \mathrm{InternetCosts}(M) \\ & \text{subject to} & \text{Policy Constraints } \mathbf{P} \\ & & \text{Constraints on } \mathrm{DelayIncrease}(M) \\ & & \text{Flow Balance Equations} \end{array}$$

Here, $\mathrm{Benefits}(M)$ are the benefits of migration to the enterprise (for instance, due to lowered operational and equipment costs), $\mathrm{InternetCosts}(M)$ is the increased communication costs since traffic between the data center and the cloud is now sent over the Internet, and $\mathrm{DelayIncrease}(M)$ is the increase in transaction delay. Since $\mathrm{DelayIncrease}(M)$ is a random variable, we impose the constraints on its statistical summary measures such as mean, variance, or percentiles. This will be discussed in more detail in §3.4. The flow balance equations guarantee that all the transactions are handled and that requests are not lost midway. We now present more details on how each of these terms may be modeled.

3.2 Flow balance equations

We represent the graph structure after migration as $G' = (V', E')$. Each node in V that corresponds to a component, say C_i, is split into two nodes, C_{iL} and C_{iR}, corresponding respectively to the servers running this application component at the local data center

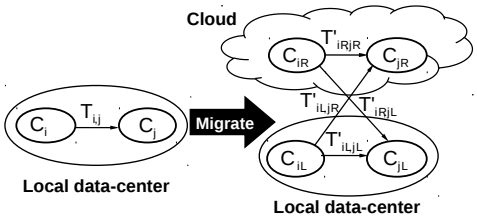

Figure 3: Partitioning requests after migration.

and at the cloud. If nodes i and j communicate in G, each replica of i is connected to each replica of j in G'. The resulting transaction matrix, denoted as $\mathbf{T}'$, must satisfy flow balance equations mentioned in §3.1. The increase in Internet communication costs and transaction delay depend on $\mathbf{T}'$. For an illustration, see Fig. 3. Depending on the policies implemented at the component servers, there are two ways to model the flow balance equations:

• **Flexible routing approach:** In this approach, the component server C_{iL} is allowed to distribute traffic differently than C_{iR} to its successor nodes. For instance, C_{iL} and C_{iR} may direct more traffic towards C_{jL} and C_{jR} respectively, so that the traffic on the Internet and the resulting costs and delays are reduced. Assume for any node i, $f_i = \frac{n_i}{N_i}$ (f_i denotes the fraction of servers to migrate). The following flow balances must be satisfied by the elements of $\mathbf{T}'$ when partitioning the original traffic $T_{i,j}$ between various replicas of component nodes i and j.

$$T_{i,j}\big(1-f_i\big) = T'_{iL,jL} + T'_{iL,jR} \quad (1)$$
$$T_{i,j}\big(f_i\big) = T'_{iR,jL} + T'_{iR,jR} \quad (2)$$
$$T_{i,j}\big(1-f_j\big) = T'_{iL,jL} + T'_{iR,jL} \quad (3)$$
$$T_{i,j}\big(f_j\big) = T'_{iL,jR} + T'_{iR,jR} \quad (4)$$
$$T'_{iL,jL}, T'_{iL,jR}, T'_{iR,jL}, T'_{iR,jR} \geq 0 \quad (5)$$

Constraints (1) and (2) model respectively that transaction requests originating at C_{iL} and C_{iR} are transferred to some servers implementing C_j. Constraints (3) and (4) model that the number of requests received at the local and remote implementations of component C_j are proportional to the number of servers at each location. Constraints involving user nodes I and O may be easily derived in similar fashion, and we omit details. If extra servers are allowed to be deployed, then we introduce variables n_{iL} and n_{iR} instead of n_i and replace the constraint $N_i f_i = n_i$ with $(1-f_i)N_i \leq n_{iL}$ and $f_i N_i \leq n_{iR}$.

• **Independent routing approach:** The flexible routing approach requires making decisions based on the location of an application component or user. Location-based routing does occur in practice - for e.g., users are typically routed to geographically close-by front-ends in many applications. However, location-based routing is harder to implement between application components in legacy applications. To handle these application constraints, we also model an approach where, the component server C_{iL} distributes traffic in the same proportions as the C_{iR} server.

Let $f(i,A) = \frac{n_i}{N_i}$ if $A = R$ and $\frac{N_i - n_i}{N_i}$ if $A = L$. Then, it is easily shown that with an independent routing approach,

$$T'_{iA,jB} = T_{i,j} f(i,A) f(j,B) \quad (6)$$

It is apparent from the definition, that the constraint (6) is more restrictive that constraints (1)-(5) and therefore admits fewer solutions. As a result the optimal solution for (MP) with (6) instead of (1)-(5) may be inferior.

3.3 Modeling Internet communication costs

The increase in Internet communication costs is easily modeled as:

$$\mathrm{Cost}_{L,I}(\mathrm{Tr}'_{L,I} - \mathrm{Tr}_{L,I}) + \mathrm{Cost}_{R,I}\,\mathrm{Tr}'_{R,I}$$

where $\mathrm{Cost}_{L,I}$ and $\mathrm{Cost}_{R,I}$ are respectively the per-unit Internet communication cost of traffic from the local and cloud data centers, $\mathrm{Tr}'_{L,I}$ ($\mathrm{Tr}_{L,I}$) and $\mathrm{Tr}'_{R,I}$ respectively denote the traffic from the local data center and the cloud to the Internet after (before) migration. We believe a linear cost model for Internet transfers is a reasonable starting point, and it matches the business model of multiple cloud providers (e.g., Amazon, Azure).

Let $\mathcal{C}=\{\mathrm{C}_i\}_{i=1}^m$, $\mathcal{C}'_L=\{\mathrm{C}_{iL}\}_{i=1}^m$, $\mathcal{C}'_R=\{\mathrm{C}_{iR}\}_{i=1}^m$, $\mathcal{L}'=\mathcal{C}_L \cup \{I\}$ and $R'=\mathcal{C}_R \cup \{O\}$. Let $i' \in V'$ ($j' \in V'$) be replicas of $i \in V$ ($j \in V$). Clearly, if $i \neq j$, $S'_{i',j'}=S_{i,j}$, otherwise $S'_{i',j'}=0$. $\mathrm{Tr}_{L,I}$ consists of traffic from the external users to components and vice-versa and can be expressed as:

$$\mathrm{Tr}_{L,I} = \sum_{i \in \mathcal{C}} \left(T_{O,i} S_{O,i} + T_{i,O} S_{i,O}\right)$$

$\mathrm{Tr}'_{L,I}$ includes the traffic from the (i) external users to the local data center and vice-versa, and (ii) traffic between local nodes (users or components) and remote components. Similarly, $\mathrm{Tr}'_{R,I}$ includes the traffic from the (i) external users to the cloud data center and vice-versa, and (ii) traffic between local nodes (users or components) and remote components. In other words,

$$\mathrm{Tr}'_{L,I} = \sum_{i \in \mathcal{L}', j \in \mathcal{R}'} \left(T'_{i,j} S'_{i,j} + T'_{j,i} S'_{j,i}\right), \mathrm{Tr}'_{R,I} = \sum_{i \in \mathcal{L}' \cup \{O\}, j \in \mathcal{C}'_R} \left(T'_{i,j} S'_{i,j} + T'_{j,i} S'_{j,i}\right).$$

3.4 Modeling increase in transaction delays

Our overall objective is to ensure the transaction delays after migration are not significantly higher than the delays prior to migration. We now discuss how constraints involving changes to mean delay, variance and percentiles may be expressed:

• **Mean delay of transactions:** Let Ω be the random variable corresponding to a user's request and the corresponding application. For each $i \in V \cup E$, let χ_i be a random variable denoting the number of times a node/directed edge is encountered per transaction, D_i be the delay experienced per encounter. Recall that, if t is the number of user requests per unit time and $T_{i,j}$ is the number of transactions on an edge $e = (i,j) \in E$, then $\frac{T_{i,j}}{t} = E[\chi_e]$. We assume that χ_i is independent of D_i, *i.e.*, the number of times i is encountered in each transaction does not influence the delay experienced per encounter. Finally, let D be the delay experienced by a user. Then, it is easily shown that:

$$E[D] = \sum_{i \in V} E[\chi_i D_i] + \sum_{e=(i,j) \in E} \left(\frac{T_{i,j}}{t} E[D_e]\right)$$

Similarly, after migration, for $i \in V' \cup E'$, we define χ'_i as the number of times i is encountered and D'_i as the delay per encounter at i. Let D' be the delay for each request after migration. As above,

$$E[D'] = \sum_{i \in V'} E\left[\chi'_i D'_i\right] + \sum_{e=(i,j) \in E'} \left(\frac{T'_{i,j}}{t} E\left[D'_e\right]\right)$$

We assume that that the time needed to process requests at a node $i \in V$ is equal to the combined time required at nodes iL and iR, i.e., $\chi'_{iL} D'_{iL} + \chi'_{iR} D'_{iR} = \chi_i D_i$. While partitioning servers after migration could potentially lead to increased service times at nodes, we assume for tractability that the service times remain unchanged.

Then, it follows that the increase in mean delay, *i.e.*, $E[D'] - E[D]$, can be written as:

$$\sum_{e=(i,j) \in E'} \left(\frac{T'_{ij}}{t} E\left[D'_e\right]\right) - \sum_{e=(i,j) \in E} \left(\frac{T_{ij}}{t} E[D_e]\right).$$

• **Variance of transaction delays:** We limit our discussion of variance to transactions that may be modeled as a path of components. We believe this is a reasonable starting point, and a majority of transactions can be modeled in this fashion. In the future, this assumption may be relaxed by modeling the flow of transactions in the network as a Markov chain.

The variance of transaction delay may be computed by considering the conditional variance of D given a path i is taken:

$$\text{VAR}[D] = E_i\left[\text{VAR}[D \mid i]\right] + \text{VAR}_i\left[E\left[D \mid i\right]\right]$$

Let DP_i be a RV corresponding to the delay of a transaction of type i which involves path P_i, and let t_i be the number of transactions involving path P_i. Then, we can show that:

$$\text{VAR}[D] = \sum_i \frac{t_i}{t}\left(\text{VAR}\left[DP_i\right] + E\left[\text{DP}_i\right]^2\right) - E[D]^2 \quad (7)$$

$\text{VAR}\left[\text{DP}_i\right]$ may be computed as the sum of the variances of the delays encountered at the nodes and links on the path ($D_i's$), assuming these random variables are independent. Applying queuing models of enterprise applications (e.g., [26]) is challenging in our context given the system after migration involves optimization variables (such as the new transaction matrix). While our evaluations indicate that the migration decisions predicted by our model work well (§5), and the applications we consider are typically overprovisioned, it would be interesting to incorporate queuing models in our framework in the future.

At first glance, it may appear that computing the variance requires detailed knowledge of the paths taken by individual transactions. Such information may be difficult to obtain if the number of possible transaction paths is large. However, we have been able to simplify Equation 7 to show that it suffices to know the number of transactions involving any pair of components or links. We believe such information can be obtained in many realistic settings, and possibly even derived from T_{ij} values for certain types of application topologies.

Percentiles of transaction delays: A constraint of optimization may be to minimize the change in percentile value of transaction delay. Accurate estimates of delay percentiles is challenging and requires detailed knowledge of delay distributions of individual random variables. Instead, we estimate the delay percentile for a particular scenario using Chebyshev's inequality. The inequality states that no more than $1/k^2$ of the values of any arbitrary distribution can be more than k standard deviations away from the mean. By estimating the mean and variance as above, and by using Chebyshev's inequality, a bound may be obtained on any percentile of transaction delay, which could then be fed into the optimization.

3.5 Modeling benefits of migration

There are several factors that can enable enterprises to reduce their costs as they migrate to the cloud. First, large cloud data centers can purchase hardware, network bandwidth, and power at much lower prices than medium-sized data-centers, and pass on these economies of scale to enterprises. Second, moving to the cloud potentially helps lower operational expenses through workforce reductions. Finally, the ability to dynamically request additional server resources to provision for peak workloads frees enterprises from the need to provision for worst-case scenarios.

In this paper, we consider a simple model for measuring benefits, which serves as a useful starting point for evaluating migration trade-offs. We assume that there are two primary classes of components: (i) compute-intensive; and (ii) storage-intensive. We assume that the benefits of migrating a single server in each class is B_c and B_s respectively. Let M_c and M_s be the total number of components in each class migrated. Then, we compute the total benefits of migration as $B_cM_c + B_sM_s$. While we assume the benefits for migrating all servers in a class is the same, the model could be easily extended to consider heterogeneity in benefits across servers in each class, which may arise for instance due to the the age of the hardware already in place in the enterprise.

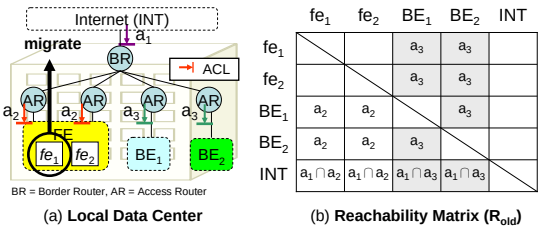

Figure 4: ACL placement in a data center and the corresponding reachability matrix if fe_1 is migrated.

Estimating the benefits per server, B_c and B_s, is in general non-trivial, and is potentially dependent on the particular enterprise and choice of cloud provider. An infrastructure-as-a-service offering like EC2 [1], for instance, might not obviate the need for database adminstrators unlike a platform-as-a-service offering such as Azure [8]. While our evaluations rely on generic cost savings estimates (e.g., [14]), we envision that in practice, more precise factors may be provided by individual enterprise managers taking site-specific considerations into account.

In this paper, we focus on the recurring costs of migration, such as server and wide-area Internet communication costs. Executing the migration process may involve one-time costs, such as the effort in acquiring model parameters, and reengineering applications for cloud deployment. Comparing one-time costs with recurring costs brings in issues such as discounting which are specific to enterprises. With appropriate discounting factors available, one-time costs can be easily incorporated in the model.

3.6 Solving the optimization problems

If a flexible routing approach (§3.2) is used, and only constraints involving changes to mean delay are considered, our optimization problem corresponds to an integer programming problem. Though integer programming problems are hard to solve, well-known tools like CPLEX [6] are available to handle them. Constraints involving variance (and percentiles) of transaction delay, or use of an independent routing approach (§3.2) lead to non-linear problem formulations. While analytically tractable solutions are difficult to obtain with non-linear optimization problems, recent theoretical advances have led to the development of tools such as BARON [20], which we leverage. While such tools are effective, scaling to large problem formulations may require tighter relaxations that constrain the search space. We do not explore these issues further in this paper, and defer them to future work.

4 Migrating reachability policies

While §3 presented a framework to help decide which servers should be migrated to the cloud, an important challenge that must be addressed is how security policies must be reconfigured. We discuss our approach to tackling this challenge in this section.

4.1 Abstraction and problem formulation

The key requirement of reachability policy migration is to ensure *correctness* – if a packet between two nodes is permitted (denied) prior to migration, it must be permitted (denied) after migration.

To aid our discussion, consider Fig. 4, which shows a data center. Consider an application with one front-end component FE, and two back-end components, BE_1, and BE_2. Assume FE has two servers fe_1 and fe_2. Assume the servers in each component

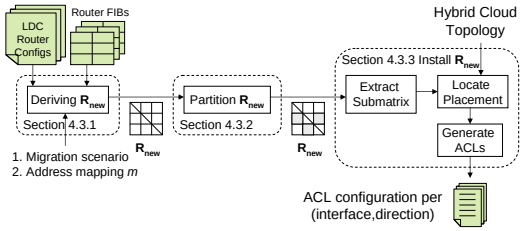

Figure 5: Overview of ACL migration algorithm.

correspond to a separate VLAN (though we note that in general severs in multiple components may be part of a VLAN). The figure also shows 3 ACLs (a_1–a_3). Consider a scenario where fe_1 is migrated to the cloud, and (possibly) assigned a new IP address.

We define an **entity** e to be an atomic unit during the migration process, i.e., all constituent servers of an entity either migrate or do not. While at the finest granularity, each application server could be considered as an entity, we prefer to model an entire VLAN as an entity when possible. As we will see, using a coarser representation when possible will ensure that our algorithm scales better. In Fig. 4, we model servers fe_1, and fe_2 as separate entities, and entire VLANs BE_1, and BE_2 as entities.

Consider a network with N entities $\{e_1, e_2, \cdots, e_N\}$. Let the **reachability matrix**, denoted by $\mathbf{R}$, be an N by N matrix, where each cell $\mathbf{R}(i,j)$ captures the *reachability set*, or the subset of packets (from the universe of all IP packets) that the network may carry from e_i to e_j [29]. Fig. 4(b) shows the reachability matrix for the scenario in Fig. 4(a). Note that we treat the Internet as an entity.

Let $\mathbf{R_{old}}$ and $\mathbf{R_{new}}$ respectively denote the reachability matrices prior to, and after a migration. Let p denote a packet typically identified by the 5-tuple (sip, dip, sp, dp, $proto$), i.e., source/destination IPs and ports, and protocol. Let m define a mapping from every IP address in the old setting to the new one, i.e., $ip_i^{new} = m(ip_i^{old})$, where ip_i^{old} and ip_i^{new} are the old and new IP addresses of e_i respectively. We define the correctness criteria below:

1. $\forall i,j$, $p \in \mathbf{R_{old}}(i,j)$ iff $p_{new} \in \mathbf{R_{new}}(i,j)$, where p=(sip,dip,sp,dp,$proto$) and p_{new}=($m(sip)$,$m(dip)$,sp,dp,$proto$)

2. $\mathbf{R_{new}}$ is invariant under failures in the new topology.

A desirable criterion when migrating reachability policies is to avoid *unwanted Internet traffic*. That is, unauthorized traffic between the local data center and the cloud must be filtered before it traverses the wide-area Internet link when possible. For instance, in Fig. 4(a), we would like traffic from fe_1 to be filtered in the cloud itself, rather than by a policy in the local data center. This criterion is important as it reduces unnecessary communication between the data centers and hence the associated costs.

4.2 ACL migration algorithm

We now present our algorithm for reconfiguring reachability policies when entities in a local data center (LDC) are migrated to a cloud data center (CDC). The migration algorithm proceeds in three phases as shown in Fig. 5: (i) $\mathbf{R_{new}}$ is derived by inferring $\mathbf{R_{old}}$ from the original network, and revising ACLs to reflect changes to address assignments; (ii) $\mathbf{R_{new}}$ is partitioned – this step helps prevent unwanted Internet traffic, as we will describe in § 4.2.3; and (iii) ACLs in each partition of $\mathbf{R_{new}}$ are installed to localize unwanted traffic while ensuring correctness. While we focus on migration from one LDC to a single CDC, the algorithm can be easily generalized to consider multiple LDCs and CDCs.

4.2.1 Granularity of policies in R

The naive approach to representing policies in $\mathbf{R}$ involves operating at the granularity of cells, i.e., computing the effective ACL corresponding to each pair of entities. In fact such an approach has been used in [24]. However, this naive approach can lead to an explosion of rules, and consequently does not scale well to large-sized networks (see §5.3.3). Many reachability policies apply to sets of entities. Representing each cell as ACL rules unrolls such a structure and hence could significantly increase the number of rules required to be installed, adding extra processing overhead which could degrade network performance after migration.

We instead adopt an alternate approach which preserves information regarding common entity pairs affected by each ACL. For each ACL a, we define its *filter domain* F_a to be the set of origin-destination (OD) entity communication pairs (i,j) (i.e., all packets from e_i to e_j) that are filtered by a. For instance, the shaded cells in Fig. 4(b) form the filter domain for ACL a_3. By maintaining the filter domain associated with each ACL, and exploiting this information, our algorithms strive to prevent a significant increase in the number of rules, as we describe in later subsections.

4.2.2 Deriving $\mathbf{R_{new}}$

The algorithm starts by inferring $\mathbf{R_{old}}$ from the LDC. The reachability policies governing traffic from e_i to e_j (i.e., $\mathbf{R_{old}}(i,j)$) contain the set of ACLs encountered along the default routing path from e_i to e_j. If no ACLs lie on the path, the cell is empty, indicating that e_i has full reachability to e_j.

The IP addresses of entities may change when they are moved to the cloud. Therefore, policies in $\mathbf{R_{old}}$ should be correctly *translated* based on the new address assignment. For example, consider Fig. 4. If fe_1 (IP address $ip_{fe_1}^{old}$) was permitted to talk to port 8000 of all servers in entity BE_1 then the same communication from $ip_{fe_1}^{new}$ to BE_1 should still be allowed after fe_1 is migrated. We derive $\mathbf{R_{new}}$ by applying a transformation function t on every ACL. The function t revises ACL rules to avoid inconsistent filtering decisions due to address reassignment. Fig. 6 shows $\mathbf{R_{new}}$ for the migration scenario in Fig. 4.

4.2.3 Partitioning $\mathbf{R_{new}}$

While traffic between a pair of entities can be filtered anywhere on their communication path without violating the correctness criteria, it is desirable to filter unwanted traffic before it enters the Internet. In particular, traffic between the LDC and the CDC should be filtered by the originating data center, rather than by the destination data center.

To achieve this, we identify for each data center DC, all OD-pairs that should be filtered at DC. We refer to this set of OD-pairs as the filter zone Z^{DC} of data center DC. Z^{DC} consists of (i) OD-pairs that originate from DC and (ii) OD-pairs that originate from the Internet and are destined to DC. For example, $\mathbf{R_{new}}$ in Fig. 6 is partitioned into Z^{LDC} (gray cells) and Z^{CDC} (dotted cells). Each partition corresponds respectively to the OD-pairs (or cells) that should be filtered by the LDC and the CDC.

4.2.4 Installing $\mathbf{R_{new}}$

For each filter zone, the relevant ACLs must be placed and the corresponding new ACL configurations must be generated. We describe the key steps below:

Submatrix Extraction: Let $F_a(DC)$ denote the filter domain of ACL a within Z^{DC}. The goal of this step is to ensure that ACL a is placed between every OD-pair $\in F_a(DC)$ in the migrated network setting. Our overall approach is to break $F_a(DC)$ into groups of cells that involve disjoint sets of source and destination entities.

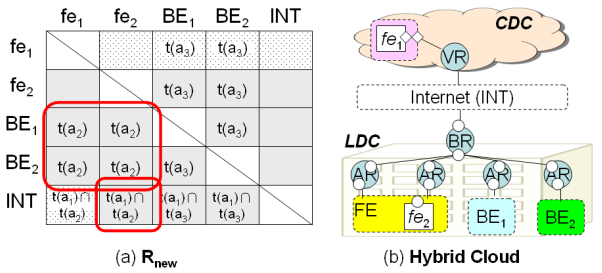

Figure 6: $\mathbf{R_{new}}$ and the hybrid cloud topology based on the migration scenario in Fig. 4.

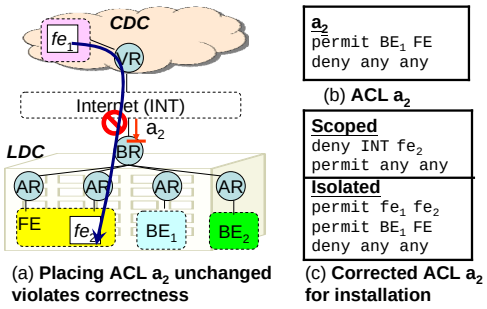

Figure 7: Applying ACL a_2 without changing its configuration inadvertently blocks allowed traffic. One of the corrected configurations should be used instead.

As we will see, having two disjoint sets of entities enables us to place ACL a on the edge-cut-set between them. We refer to such a place-able group of cells as a *submatrix*. For scaling reasons, we prefer to keep the number of submatrices extracted for $F_a(DC)$ small. Formally, let $F_a(DC)$ denote the filter domain of ACL a within Z^{DC}. We extract a minimal set of submatrices covering all OD-pairs $\in F_a(DC)$ by repeatedly removing the largest submatrix from $F_a(DC)$ until no OD-pair remains. We denote the kth submatrix as $F_a(DC,k)$. For example, Fig. 6(a) highlights the two submatrices (circled) extracted for $F_{a_2}(\text{LDC})$.

Locating Placement: For each submatrix $F_a(DC,k)$ extracted, the corresponding ACL a needs to be placed inside DC, such that packets going from the source entities to the destination entities will encounter an instance of the ACL no matter which physical path they take. To achieve this, we place the ACL along an edge-cut-set between the associated sets of source and destination entities in the new topology. An edge-cut-set consists of a set of *placement locations* $\{l\}$, each specifying an interface and the direction of traffic (inbound or outbound) to which an ACL placed on that interface applies.

In general, each cloud provider has its own infrastructure, which influences where ACLs can be placed in the CDC. For instance, some cloud providers may grant cloud users the ability to perform VM-level filtering [1], or offer proprietary techniques to specify ACLs for groups of VMs [3,10]. For example, Fig. 6 shows the hybrid cloud topology based on the migration scenario in Fig. 4. Eligible placement locations in the LDC and the CDC are marked by circles and diamonds, respectively. Note that the virtual router (VR) in the cloud is not eligible for placement.

To determine the placement locations, we compute the minimum edge-cut-set, only allowing links incident to eligible placement locations to be part of the edge-cut-set. We achieve this using polynomial algorithms for finding the minimum cut in a flow network [16]. If both ends of a link are eligible locations, we place ACL a on the interface closer to the source entities.

Generating ACL Configuration: Finally, the appropriate ACL configurations must be generated for each placement location. Directly installing each ACL may accidentally filter other traffic. For example, after migration, assume ACL a_2 is placed as shown in Fig. 7(a), to block traffic from INT to fe_2. Fig. 7(b) depicts the rules of ACL a_2 prior to migration. If a_2 is installed unaltered as in Fig. 7(a), traffic from fe_1 to fe_2 is inadvertently blocked, violating the correctness criteria. Note that this was not an issue prior to migration (see Fig. 4), as traffic from fe_1 to fe_2 did not encounter a_2 in that scenario.

Consider the placement of submatrix $F_a(DC,k)$ of ACL a. We introduce two methods, *scoping* and *isolation*, for generating the correct ACL configuration $a(l)$ for each location l on which ACL a is placed. We define *traffic domain* D_l at location l to be all possible OD-pairs that can traverse through l. Scoping ensures that $a(l)$ only filters traffic $\in D_l \cap F_a(DC,k)$. Isolation ensures that $D_l \cap \overline{F_a}(DC)$ is allowed by ACL a. For example, the scoped and isolated versions of ACL a_2 are shown in Fig. 7(c). We compute both the scoped and isolated versions of ACL a based on location l, pick $a(l)$ as the smaller of the two, and merge it with any co-located ACL to generate the final ACL configuration.

5 Evaluation

This section presents results evaluating the importance and effectiveness of our model in planning hybrid cloud layouts. Our evaluations were conducted using (i) a real application made available as part of the Windows Azure SDK (§5.1), and (ii) a real Enterprise Resource Planning (ERP) application deployed in a large campus network (§5.2). We present evaluations of the ACL migration algorithm in §5.3 based on security policies from the campus network.

5.1 Planned migration of a simple enterprise application

Our evaluations were conducted by deploying an application provided as part of the Windows Azure SDK on the Windows Azure cloud platform. We describe the application data flow, the setup of our cloud test-bed, and discuss how we instrumented the application to obtain parameters needed by our model. We then present results evaluating the change in response time when the application is deployed across multiple data centers in a configuration recommended by our planned migration approach. These results help us experimentally validate the effectiveness of our model in meeting constraints on changes in application response time.

Application abstraction and data-flow: The application that we refer to as *Thumbnail* involves a user uploading a picture to a server. The server creates a thumbnail version of the picture and returns it to the user. The application consists of three components - a web front-end (FE), a Windows Azure service for temporary storage known as a Blob (BE), and a worker role (BL). Fig. 8 illustrates the data-flow. A user initiates a transaction by uploading a picture and sending a request to FE (t0). The FE pushes a notification message into a queue while writing the image to the BE (t1-a and t1-b). Note that since these operations occur in parallel and the time taken for t1-b dominates, we exclude the queue from our abstraction. The BL reads the message from the queue, gets the image from the BE (t2), creates a thumbnail, and stores the thumbnail in BE (t3). Once the thumbnail is generated, the FE retrieves the thumbnail (t4) and sends a response page, along with the thumbnail, back to the user (t5).

Cloud test-bed setup: While the original application was implemented to only run on one data center, we have revised it so each component may span multiple data centers. We create a setup involving two different Azure data centers located in geographically different locations - one in north-central United States (DC_N), and

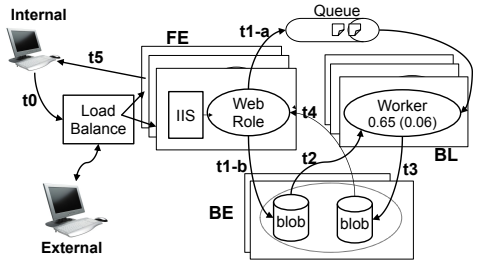

Figure 8: Data flow of thumbnail application.

the other in south-central United States (DC_S). We view DC_N and DC_S respectively as the local and cloud data centers. We picked a host (I) located in geographical proximity to DC_N as seeing performance representative of an internal enterprise user. We picked about 20 Planetlab hosts (O) scattered around the United States, and assumed they corresponded to external users.

Deriving model parameters: We have instrumented the application to measure transaction sizes, component service times and various communication delays. Using the setup above, we run the application over 50 times by uploading pictures from a typical user album. The mean (stddev) time for I to upload a picture to DC_N and DC_S were measured to be 2966 (742) and 4681 (818) msec, while the similar values from O to DC_N and DC_S were 3422 (619), and 3494 (498) msec. The mean (stddev) time to download the thumbnail from both clouds was 505 (208) msec for I and 455 (244) msec for O. Before migration, the entire application resides in the same data center. The mean (stddev) delay is 718 (169) msec from FE to BE, 251 (160) msec from BE to BL, and 66 (16) msec from BL to BE. The mean (stddev) of the service time of BL is 655 (60) msec. The service time of FE and BE is negligibly small. We also measured the transfer delays of the original images and the thumbnails between the clouds over multiple runs. The mean (stddev) transfer time of the original images and thumbnails between the two data centers was 2044 (161) msec and 96 (17) msec respectively. Finally, the transaction size between each component pair was the average size of the image or the thumbnail. These values were 1500 KBytes and 4 Kbytes, respectively.

Modeling migration benefits and communication costs: We assume that migrating servers to the cloud can reduce costs by a factor of 7 for compute-class servers, and 5 for storage-class servers, as suggested in [14]. We leverage the Amazon EC2 cloud pricing [1] to calculate the cost of running a server in the cloud. We consider a scenario where a total storage space of 1TB is required, and 1000 I/O transactions per second are involved. The resulting benefits of migrating a compute-class server is $1577 per year, and the benefits of migrating a storage-class server is $17280 per year. Finally, based on [1], we assume that it costs $170 for exchanging 1TB of data with the cloud.

Migration strategies recommended by our model: Table 1 summarizes the results obtained using our model with the flexible routing approach for a scenario where (i) 80% of users are internal, and the rest are external; and (ii) there are four servers in each component. Each row (column) corresponds to a constraint on the mean delay (variance). Each cell shows the migration strategy recommended by our model, as well as the *yearly* savings in US dollars. For example, a setting (V=150% and D=110%) means a variance bound of 150% of the original variance, and a mean delay bound of 110% of the original mean delay. For the same setting, 1 FE server, 3 BL servers, and 2 BE servers should be migrated for a maximum yearly savings of $36367. We have run our algorithm for other user mixes, but omit the results for lack of space.

We make several observations. First, in most cases, more BL and BE servers are migrated than FE servers. Further, the number of BL and BE servers migrated is often the same. This is because (i) moving a BE server achieves more benefits than moving FE servers; and (ii) by moving the same number of BL servers as BE servers, all pictures that were sent to the BE servers can be processed by BL servers in the same location. Second, variance plays an important role in the recommendations. For instance, for a delay bound of 110% (row 2), the best migration strategy varies for different constraints on the variance. Third, we checked how the traffic was routed between components, and confirmed that our approach routed traffic intelligently. For example, external user transactions were always routed to the cloud to the extent possible.

D \ V	125%	150%	175%	no bound
105%	1/1/1, $20024	1/1/1, $20024	1/1/1, $20024	1/1/1, $20024
110%	1/1/1, $20024	1/3/2, $36367	1/2/2, $36836	2/2/2, $38413
150%	1/1/1, $20024	1/3/3, $53647	1/3/3, $53647	1/3/3, $53647
200%	1/1/1, $20024	1/3/3, $53647	2/3/3, $55224	3/3/3, $56801

Table 1: Recommendations of planned migration approach for the Thumbnail application. The dollar amounts shown are savings per year.

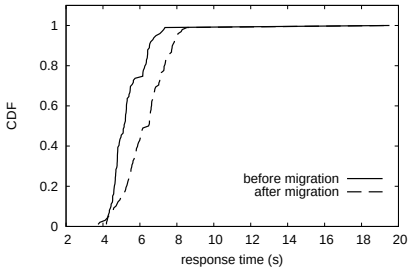

Figure 9: Validating model recommendations: CDF of user response time (in seconds) before and after migration.

Validating recommendations through cloud deployment: We deployed the recommended migration strategy for the scenario corresponding to 80% internal users, with constraints of up to 10% increase in mean delay and 50% in variance. The strategy recommends migrating one FE, two BE, and three BL servers. Further, as generated by the flexible routing approach, all requests from external users followed the path $\langle FE_R, BE_R, BL_R \rangle$. 6.25% of local requests followed the path $\langle FE_R, BE_R, BL_R \rangle$, whereas the remaining requests were split evenly among paths $\langle FE_L, BE_L, BL_L \rangle$, $\langle FE_L, BE_L, BL_R \rangle$, and $\langle FE_L, BE_R, BL_R \rangle$. Here FE_L and FE_R respectively denote the local and remote components of FE, and similar notations are used for other components.

Fig. 9 presents a CDF of user response times obtained using the cloud test-bed for the scenarios prior to and after migration. The values were obtained over 100 user transactions, with internal and external users simulated using a host in geographical proximity to the data center, and using Planetlab hosts as described before. While response times after migration increase, as is expected, the increase is still within acceptable limits. For instance, the 90%ile of response times increased from 6.5 to 7.7 seconds, while the 99%ile increased from 7.4 to 8.4 seconds. We observed an increase of 17% in mean delay, and 12% in variance. At 5% level of significance, a *t*-test of difference in the expected response times did not provide sufficient evidence to conclude that the mean response time had increased more than 10% after migration.

5.2 Planning migration of a campus ERP application

We next present a model of a real Enterprise Resource Planning (ERP) application used in a large university with tens of thousands of students, and several thousand faculty and staff. We use the application as a case study to illustrate the benefits of a hybrid ap-

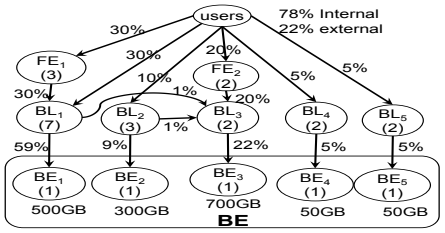

Figure 10: The ERP application in a large university.

proach, and to show that our model accounts for key factors that may impact migration decisions.

Modeling a deployed ERP application: Fig. 10 presents a model of the application, obtained after discussions with the campus network operators. The top most node represents users (both internal and external). About 78% of the transactions are generated by users internal to the enterprise, and the rest by external users. External users include those in satellite campuses of the main university campus. Every other node corresponds to an application component, and is annotated with the number of servers in that component. There are two front-end components (FE_1, FE_2), and five business-logic components (BL_1–BL_5). The components in the BL layer differ in the transactions they handle - for instance, one component handles financial transactions, while another handles supply relations related transactions. The application has five databases, each of which could be modeled as a back-end component (BE_1–BE_5, collectively referred to as BE). Each database has a size as shown in the figure. Each edge indicates that the corresponding pair of nodes can communicate, and is annotated with the percentage of transactions in the system that use that edge.

The figure highlights interesting departures from conventional text-book application design. First, while about half the transactions from users are directed to the front-end components (these users are called *thin clients*), the remaining transactions directly reach the business logic components (generated by *thick clients*). Second, we observe communication between components in the BL layer. For instance, once a purchase order is finalized, a BL component corresponding to supply relations invokes a BL component corresponding to financial transactions to ensure the funds get committed. In fact, all pairs of BL components interact in practice, though many interacting links carry a negligible fraction of transactions and are not shown in Fig. 10. Finally, while in general the traffic between components mirrored the number of transactions between them, the data warehouse component (BL_3) accounted for 70% of traffic between the BL layer and the back-end.

Inferring model parameters: We derived parameters that our model needs by conducting end-to-end measurements of typical user requests (such as downloading salary statements), and estimating individual component and link communication times. Our measurements indicated typical user requests involved uploads of about two KBytes, downloads of about 13.4 KBytes, end-to-end response times of 1400 msec, and file download times of 202 msec. From these measurements, we inferred communication delays on other links assuming they were proportionally scaled from the request upload and file download times. The difference between the end-to-end response times and the link communication delays was assumed to be node service times, and we assumed 90% of the service time was spent in servers in the BL layer, as these were most compute-intensive. We also measured the upload and download times of similar-sized files to the Windows Azure cloud, to estimate communication delays with the cloud. The same values were used to estimate communication delays between local and migrated components, and delays related to external users. We assumed 10 transactions per second on average. Finally, we used the same values for migration benefits and communication costs as in §5.1.

Delay Bound	Yearly Savings	Recommended Components to Migrate		
		FE	BL	BE
115% w/ policy	\$14,102	$FE_1(1)$	$BL_1(2)$, BL_2, BL_4, BL_5	—
115%	\$37,769	$FE_1(1)$, FE_2	$BL_1(1)$, BL_2, BL_3, BL_4, BL_5	BE_2, BE_3, BE_4, BE_5
110%	\$27,789	FE_2	$BL_2(1)$, BL_3, BL_5	BE_2, BE_3, BE_5
120%	\$43,592	$FE_1(1)$, FE_2	$BL_1(2)$, BL_2, BL_3, BL_5	BE_1, BE_2, BE_3, BE_5
130%	\$57,763	migrate all components in full		

Table 2: Recommendations of the planned approach for ERP application.

Recommendations from model: Table 2 presents results produced by our model with the flexible routing approach. Each row corresponds to a particular bound on mean delay. For each delay bound, we show the yearly savings, and the components in each tier that should be migrated. If a component should only be migrated partially, we also show in brackets the number of servers to migrate. For example, for a delay bound of 110%, only one of the three servers of BL_2 are migrated, while BL_3 and BL_5 are migrated fully. The first row is a special case where the operator specifies a policy that no database (i.e., BE_1–BE_5) can be migrated. For all other rows, all components can be migrated.

Table 2 illustrates at least three scenarios where the hybrid approach could be useful. First, when there are policy restrictions on migration (first row), the hybrid approach can still identify less sensitive components to migrate, leading to savings while meeting the delay bound. Second, migrating the entire application is feasible only when the delay bound is set at 130% or higher (row 5). This is because all transactions from internal users must be sent out to the cloud and then back to the enterprise. In contrast, a hybrid approach can achieve significant savings even at lower delay bounds, by migrating a subset of components. Third, full migration of components such as FE_1 and BL_1 can have a substantial impact on delay, as these components are used by a large fraction of transactions. A hybrid approach allows for partial migration of these components (rows 2–4). Here, the subset of servers moved to the cloud are mainly used by the external users, and hence the increase in mean delay is small.

Table 2 also shows that the interaction between components plays a critical role in planning decisions. For example, for all rows, when FE_2 is migrated, BL_3 and BE_3 are also migrated. Doing so enables the transactions sent to FE_2 to be processed entirely in the cloud, and prevents them from being sent back and forth between the local data center and the cloud.

Sensitivity to model parameters: We have studied how the optimal migration strategy is impacted by varying model parameters, in particular, the estimated benefits of migrating individual CPU and storage servers. We summarize some of the key insights that we have gained. First, for some delay bounds, the optimal migration strategy dominates all other feasible solutions in that it moves more CPU and storage servers than any other approach. In such cases, the optimal migration strategy does not depend on the benefit estimates. For instance, for any delay bound above 130%, the best approach is to migrate the entire application regardless of benefit estimates. Second, the benefit estimates impact the strategy if there are multiple feasible approaches to realize the delay bound, none of which dominate the others. For example, a delay bound of 120% could be met by either migrating component BE1 (a large database), or components BL4 and BE4 (2 CPU servers and a much smaller database). While Table 2 recommended that BE1 be moved, our results show that components BL4 and BE4 should be moved instead if the CPU benefits were much higher. Third, the

Firewall Contexts	No.
Admin. Network for misc. servers, e.g., storage/backup servers	4
Admin. Network for production/non-production academic systems	2
Admin. Network for production/non-production database	2
Admin. Network for production/non-production ERP	2
Admin. Network for production/non-production web services	2
Admin. Network for production/non-production Windows/UNIX systems	6
Internal/External DMZ	2
Misc. servers/applications	6
Oracle/SQL Database Servers	3
Production/Non-production academic systems	2
Production/Non-production Active Directory	2
Production/Non-production ERP	2
Production/Non-production Windows/UNIX systems	5
Total	40

Table 3: Types of firewall contexts in the campus data center.

ACL	Rules	Default
a_1	1. deny: all but external hosts with private addresses	permit
a_2	1. permit: monitoring component (BL_5)→FE_1 and FE_2	deny
	2. permit: any→HTTP & HTTPS ports on FE_1 and FE_2	
a_3	1. permit: monitoring component (BL_5)→BL_1, BL_2, and BL_3	deny
	2. permit: BL_4→TCP port $p1$ on BL_1, BL_2, and BL_3	
	3. permit: FE_1→TCP port $p2$ on BL_1	
	4. permit: FE_2→TCP port $p2$ on BL_3	
	5. permit: external/campus users→TCP port $p3$ on BL_1 and BL_2	
a_4	1. permit: monitoring component (BL_5)→BL_4	deny
	2. permit: BL_1, BL_2, and BL_3→TCP port $p1$ on BL_4	
	3. permit: external/campus users→TCP port $p3$ on BL_4	
a_5	1. permit: external/campus administrators→SSH port on BL_5	deny
a_7	1. permit: monitoring component (BL_5)→BE	deny
	2. permit: all BL components to MySQL port on BE	

Table 4: ACL configurations before migration.

relative size of transactions between different components may determine the optimal strategy independent of benefit estimates. Note that transaction sizes determine (i) the delay between the components and consequently total application response time; and (ii) the wide-area communication costs. For instance, if the size of transactions between the BL and BE tier is larger than the size of transactions between other tiers, migrating a BE component alone is not as beneficial as migrating both the BE component and the most frequently interacting BL component, regardless of benefit estimates. Finally, we have explored the sensitivity of our results to the costs of Internet communication. For most delay bound settings, the recommended strategy does not change even if costs go up by a factor of 10, but with higher factors, fewer components should be migrated. We omit further details for lack of space, and defer studies on a wider range of applications to future work.

5.3 Migrating security policies

In this section, we demonstrate the benefit of systematic migration of ACLs by applying our algorithms to a concrete migration scenario and evaluating how well our approach performs and scales on a large-scale campus network.

5.3.1 Security policies in operational data centers

We present a high-level characterization of ACL usage in two operational data centers, DC_1 and DC_2, owned by a large-scale campus network. The data centers offer a variety of services such as student self-services, finance tools, employee training, etc., to campus and external users.

Overall, hundreds of servers are logically partitioned in 40 server VLANs (35 in DC_1 and 5 in DC_2). Servers in the same VLAN have similar functionalities and share similar patterns when communicating with nodes outside of the VLAN. Every VLAN has its ingress and egress traffic protected by a *firewall context*, which consists of a pair of ACLs (one per direction) placed at the interface of the VLAN's access router. By default, all ingress/egress traffic to/from a VLAN is denied/allowed. Any incoming traffic destined to servers in a VLAN needs to be *explicitly* allowed by adding appropriate `permit` ingress rules.

Table 3 summarizes the types of firewall contexts and the number of contexts of each type. The contexts contain diverse reachability policies for application servers, database and file servers, portal servers and application servers for external access, and non-production testing networks. Nearly half of the contexts are administration contexts, which are accessible for authorized administrators to manage servers. Fig. 11 shows a CDF of the number of ingress/egress ACL rules across all contexts. The size of each ACL ranges from 0 to 208 rules. Migration of reachability policies of such complexity and scale is daunting, further strengthening the case for an automated approach like ours.

5.3.2 Case study: migrating ERP application

We evaluate the effectiveness of our ACL migration algorithm in the context of the ERP application introduced in §5.3.2. We obtained the ACL rules relevant to the application from the campus operators. We computed the new ACL configurations for the migration scenario corresponding to the first row of Table 2. In this scenario, all servers in the components BL_2, BL_4, and BL_5, one server in FE_1 and two servers in BL_1 are migrated. All servers for the ERP application are located in DC_1 and are physically arranged as shown in Fig. 12. $FE_{1,1}$–$FE_{1,3}$ denote the individual servers of component FE_1, and a similar notation is used for other servers. All servers in the FE and the BE tier are placed in VLAN V1 and VLAN V7 respectively. Servers in the BL tier are placed in VLANs V2, V4 and V5.

Fig. 12 shows that six different ACLs are placed on nine router interfaces to filter traffic as per operator goals. Table 4 shows the rules associated with each ACL. A packet not matching any of the rules takes the default action. The policies closely match the communication pattern in Fig. 10. For example, a_3 permits access from the monitoring component BL_5 to components BL_1, BL_2 and BL_3 (rule 1), and from BL_4 to one port on the same destinations (rule 2). a_3 also permits traffic from FE_1 to a port on BL_1 (rule 3), from FE_2 to a port on BL_3 (rule 4), and from external/campus users to a port on BL_1 and BL_2 (rule 5).

Fig. 13 shows the new ACL placement after migration, assuming techniques for applying ACLs to groups of VMs are available in the cloud. After migration, there are a total of 13 ACLs since placement is done separately for each data center. Each ACL before migration may be placed in multiple locations, contributing to rules in one or more ACLs after migration. For example, ACL a_3 contributes to rules in ACLs r_1, r_3, r_5, r_7, r_{10}, and r_{11}. Each new ACL is generated by merging co-located ACLs, e.g., a_2 and a_3 are merged to produce r_5. Table 5 shows the configuration of a subset of these new ACLs that inherit rules from a_3.

We use a_3 to illustrate two key properties: correctness and filtering of unwanted Internet traffic. First, the new ACLs correctly inherit policies from a_3. For example, a_3's rules 3 and 4 are accounted for by r_3 (rules 1–3), r_7 (rule 1), and r_{10} (rule 3). Second, the new ACLs are placed at appropriate locations. a_3 was originally placed on the outbound router interface facing VLAN V2 to filter *all* traffic to the 3 BL components. After migration, $BL_{1,1-2}$ and BL_2 are moved to the cloud. To block unwanted traffic from DC_1 to these migrated servers before it reaches the Internet, we move a_3 to the inbound router interfaces facing VLANs V1 and V7, producing the new ACLs r_3 and r_5. Clearly, correct and beneficial ACL migration is tedious if done manually without using our algorithm.

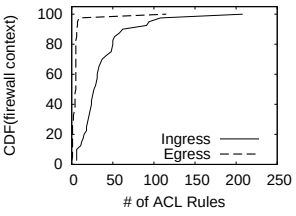

Figure 11: Ingress/Egress ACL sizes per firewall context.

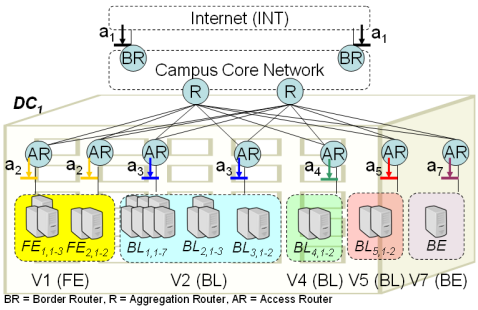

Figure 12: ACL placement for the ERP application in DC_1.

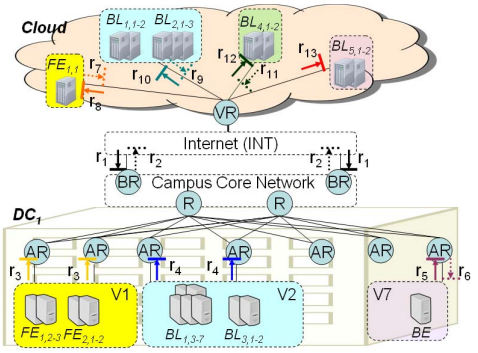

Figure 13: ACL placement after migrating the ERP application from DC_1 to the cloud.

ACL	Rules	Default
r_3	1. permit: $FE_{1,2-3}$→TCP port $p2$ on BL_1	permit
	2. permit: FE_2→TCP port $p2$ on BL_3	
	3. deny: any→BL_1, BL_2, and BL_3	
r_5	1. deny: any→BL_1, BL_2, and BL_3	permit
	2. permit: any→HTTP & HTTPS ports on FE_1 and FE_2	
	3. deny: any→FE_1 and FE_2	
r_7	1. permit: $FE_{1,1}$→TCP port $p2$ on BL_1	permit
	2. deny: $FE_{1,1}$→BL_1, BL_2, BL_3, and BE	
r_{10}	1. permit: $BL_{1,3-7}$, BL_3, BL_5→any	deny
	2. permit: BL_4→TCP port $p1$ on any server	
	3. permit: $FE_{1,1-3}$→TCP port $p2$ on $BL_{1,1-2}$	
	4. permit: external users→TCP port $p3$ on any server	

Table 5: Subset of ACLs inheriting rules from a_3 after migration.

5.3.3 Performance and scalability

To evaluate the scalability of our ACL migration algorithm, we ran it on the reachability policies of the entire campus network, which is a distinct rule-set from that described in § 5.3.1 corresponding to policies in the campus data-center. The network consists of 700 VLANs and 212 ACLs. The ACLs are enforced on 341 locations across 63 routers, and range in size from 2 to 300 rules. 24% of the routers have 200+ rules. We derived the reachability matrix and reinstalled it under the same network setting. Our algorithm took 4 minutes to run on a dual-core Intel Xeon 2 2.6GHz system with 8GB RAM using a Java implementation of the algorithm.

The total number of ACL rules across all routers in the network is 7889. If a naive approach of computing and placing each cell of the reachability matrix was used (§4.2.1), the total number of rules grows by two orders of magnitude (570521). In contrast, if our approach that works at a coarser granularity is employed, the number of rules is 7952; only 63 more than the original network. The slight growth in rules is due to an inconsistent policy in the original network and we omit details for lack of space. Overall, these results show that our approach can scale well to large networks.

6 Related work

Many challenges must be addressed before enterprises can embrace the benefits of cloud computing. These include lock-in to specific providers, lack of scalable storage, risks in fate-sharing with others in shared infrastructure, service availability, and problem resolution in the cloud [14,23]. Enterprises are also concerned about how they can stream-line the deployment of their complex services in the cloud [23]. Our paper takes a step in this direction by developing a framework for deciding what to migrate to the cloud such that enterprises can realize a benefit. Maintaining the security and privacy of data once migrated to the cloud is a challenge [14, 28], and has started receiving attention from the community [19]. Existing solutions [2, 28] propose extending the enterprises' network into the cloud using a VPN in order to isolate the enterprises' instances in the cloud. Using such a model, our security framework can be directly leveraged to ensure that the security policies in the data center extend to the services on the VPN in the cloud.

Previous work in the analysis of multi-tier applications focused on developing queuing models of applications that can be used to estimate mean response times [26]. A dynamic provisioning technique for such applications that employs a queuing model is presented in [27]. In contrast, we use an optimization framework to identify application components to migrate to the cloud in order to realize the maximum benefit, and consider variance in addition to mean response times. We assume direct measurements of response time distributions of the overall system, and individual components prior to migration are available. These already account for queuing considerations before migration. Our model implicitly assumes the queuing structure does not change after migration. Incorporating queuing models to account for such a change within our optimization framework is an interesting direction for future research.

Algorithms for placing security policies when deploying new enterprise networks are presented in [24]. In contrast, we focus on unique issues in migrating existing enterprise applications to the cloud. In [24], it is assumed that the reachability matrix information is defined at the granularity of individual cells. This approach is impractical in practice as it involves an explosion in the number of ACL rules (§4.2.1, §5.3.3). In contrast, our approach provides an efficient intermediate representation and operates at a coarser granularity, ensuring better scaling with large networks.

7 Discussion and Open Issues

We discuss key aspects of our work, and open issues:

Model enhancements: While our paper helps better understand cloud migration trade-offs, it is only a start. An important future direction is understanding the impact of migration on application reliability, given the high costs of down-time for enterprise applications [18]. While on the one hand, components in the cloud run at lower SLAs than components in the enterprise [23], migration increases the number of fault domains in which the application operates, and so has the potential to increase reliability. In this paper, we have framed the problem as one of deciding how many servers to migrate to the cloud, and focused on a two location model (local and cloud data-center). In the future, it would be interesting to generalize this to models that allow any number of servers to be instantiated in the local and cloud data-centers, and allow for multiple cloud locations. Finally, it may be interesting to extend our cost and latency models to consider middle-boxes deployed in enterprises. WAN optimizers [9] could potentially reduce Inter-

net communication costs, if deployed at both the local and cloud data-centers. Encryption of data over the public Internet may incur additional CPU costs, as well as higher latencies, though we note the increase in latency is relatively small.

Handling dynamic variations in workload: An important benefit of hybrid architectures that we have not explored in this paper is their potential to help handle peaks in workload. In particular, the local data-center could be provisioned with enough server capacity to handle typical workloads, while cloud resources could be invoked as needed to deal with peaks. That said, our modeling approach could potentially help in planning layouts that can deal with dynamic workload variations. One approach is to use the model to determine the appropriate configurations for a variety of estimated workloads, and base the final configuration on the expected probabilities of each workload. Another approach is to use our model periodically as workloads change over time, to determine if a change in placement is required. We defer more detailed investigation of these issues to future work.

Executing migrations: This paper focuses on the questions of whether migration is beneficial at all, and how to determine what to migrate. Executing a migration itself poses several challenges such as identifying dependencies and changing application server configurations, minimizing application down time, and efficiently copying large databases while synchronizing local and remote replicas. While technologies such as live migration can be used to minimize service disruption, extending such technologies to wide-area environments is an area that needs more research.

Obtaining model parameters: Prior to migration planning, we need to perform application discovery to obtain essential input parameters such as application dependencies, component response times, and traffic exchanged between components. Many research tools and commercial products can be used to obtain application dependencies either by inference from network communication patterns [13], or by analyzing application-level configurations [7, 21]. Transaction counters and service response time measurements for each component are widely available on enterprise servers today. Most enterprise software packages provide embedded performance monitoring capabilities that can be enabled on the servers themselves to track and report these performance numbers [5, 12]. Finally, inaccuracies in estimates of model parameters could be dealt with by running the model with multiple sets of inputs and choosing a conservative plan to ensure application response times are met.

8 Conclusion

In this paper, we have made two contributions. First, we have shown (i) the potential benefits of hybrid cloud deployments of enterprise applications compared to "all or nothing" migrations; and (ii) the importance and feasibility of a planned approach to making migration decisions. Second, we have shown the feasibility of automatic and assurable reconfiguration of reachability policies as enterprise applications are migrated to hybrid cloud models. We have validated our algorithms using a campus ERP application, Azure-based cloud deployments, and router configurations of a large campus network. Our work is an important but first step. In the future, we hope to gain experience with our approach using a wider range of real enterprise applications, explore the predictive power of our model under a wider range of settings, and adapt our approach to dynamic variations in workload.

9 Acknowledgments

We thank Brad Devine, William Harshbarger, Michael Schulte, Kitch Spicer and others in the Information Technology Department at Purdue (ITaP) for providing access to the data, and for their time. We thank our shepherd, Jeff Mogul, and the anonymous reviewers for their feedback which helped substantially improve the quality of the paper. This work was supported in part by NSF grants CNS-0721488 and Career-0953622.

10 References

[1] Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2/.
[2] Amazon Virtual Private Cloud. http://aws.amazon.com/vpc/.
[3] Amazon Web Services: Overview of Security Processes, White Paper. http://aws.amazon.com/security.
[4] Animoto - Scaling Through Viral Growth. http://aws.typepad.com/aws/2008/04/animoto---scali.html.
[5] DB2 for z/OS Performance Monitoring and Tuning Guide. http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.perf/dsnpfk17.pdf.
[6] IBM ILOG CPLEX. http://www-01.ibm.com/software/integration/optimization/cplex/.
[7] IBM Service Management Tivoli Application Dependency Discovery Manager Software. http://www-01.ibm.com/software/tivoli/products/taddm/.
[8] Microsoft Windows Azure. http://www.microsoft.com/windowsazure/.
[9] Riverbed Technology. http://www.riverbed.com/.
[10] SQL Azure Firewall. http://msdn.microsoft.com/en-us/library/ee621782.aspx.
[11] The Case Against Cloud Computing. http://www.cio.com/article/477473/.
[12] Websphere application server performance monitoring infrastructure (pmi). http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/cprf_pmi_arch.html.
[13] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance Debugging for Distributed Systems of Black Boxes. In *Proc. SOSP*, 2003.
[14] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.
[15] Arthur Cole. The Future Belongs to Hybrid Clouds. http://www.itbusinessedge.com/cm/blogs/cole/the-future-belongs-to-hybrid-clouds/?cs=30482.
[16] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
[17] D. Gottfrid. The New York Times Archives + Amazon Web Services = TimesMachine. http://open.blogs.nytimes.com/2008/05/21/.
[18] J. Hamilton. The Cost of Latency. http://perspectives.mvdirona.com/2009/10/31/TheCostOfLatency.aspx, Oct. 2009.
[19] L. E. Li, M. F. Nowlan, C. Tian, Y. R. Yang, and M. Zhang. Mosaic: Policy Homomorphic Network Extension. Technical Report YALEU/DCS/TR-1427, CS Department, Yale University, Feb 2010.
[20] M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004.
[21] K. Magoutis, M. Devarakonda, and K. Muniswamy-Reddy. Galapagos: Automatically Discovering Application-Data Relationships in Networked Systems. In *Proc. IM*, 2007.
[22] M. O'Neill. Connecting to the cloud. http://www.ibm.com/developerworks/library/x-cloudpt1/.
[23] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, and C. Dorai. Are Clouds Ready for Large Distributed Applications? In *Proc. SOSP LADIS Workshop*, 2009.
[24] Y.-W. E. Sung, S. G. Rao, G. G. Xie, and D. A. Maltz. Towards systematic design of enterprise networks. In *Proc. CoNEXT*, 2008.
[25] Symantec. 2010 State of the Data Center Global Data. http://www.symantec.com/content/en/us/about/media/pdfs/Symantec_DataCenter10_Report_Global.pdf.
[26] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proc. SIGMETRICS*, 2005.
[27] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic Provisioning of Multi-tier Internet Applications. In *Proc. ICAC*, 2005.
[28] T. Wood, P. Shenoy, A. Gerber, K. Ramakrishnan, and J. V. der Merwe. The Case for Enterprise-Ready Virtual Private Clouds. In *Proc. HotCloud Workshop*, 2009.
[29] G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of IP networks. In *Proc. INFOCOM*, 2005.