RC25438 (WAT1401-010) January 2, 2014
Computer Science

IBM Research Report

On the Challenges and Solutions for Migrating Legacy Distributed Applications into Cloud

Chunqiang Tang, Byung Chul Tak, Long Wang, Hai Huang, Salman Baset
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
USA

On the Challenges and Solutions for Migrating Legacy Distributed Applications into Cloud

Chunqiang Tang, Byung Chul Tak, Long Wang, Hai Huang, and Salman Baset
IBM T.J. Watson Research Center
1101 Kitchawan Rd, Yorktown Heights, NY

ABSTRACT

As the adoption of cloud continues to grow, organizations are faced with the challenge of migrating legacy applications into the cloud, or at least evaluating whether a migration is worthwhile. The simple solution of a fresh application reinstallation in the cloud is often labor intensive and has a high risk of running into unknown problems. On the other hand, importing already-installed images requires intimate application knowledge to perform reconfiguration, e.g., pointing one component to the new IP address of another component. The purpose of the paper is to present the migration problem to the research community. We devise a methodology for defining the migration solution space, and propose two techniques, *AppCloak* and *PseudoApp*, that hopefully can help advance the state-of-the-art of migration.

1. INTRODUCTION

Cloud computing is a powerful force that has been reshaping the landscape of the IT industry. Many organizations have either already embraced cloud or been actively exploring it to determine whether it is a good fit for their workloads. In cloud research, much attention has been given to finding new killer applications, e.g., data intensive analytics with MapReduce. However, how to provide a smooth transition path from today's technology to a cloud environment has been mostly neglected in the literature. In this paper we explore the challenges and potential solutions in migrating legacy distributed applications to Infrastructure-as-a-Service (IaaS) clouds.

1.1 Challenges

One natural way of migrating an application to cloud is to reinstall all application components in the cloud from scratch. Installing and configuring a complex enterprise application is often time consuming and error prone, thus, there is a significant risk of encountering unknown issues and missing the project deadline.[1] Another even bigger challenge is that a legacy application may have gone through many undocumented changes over a long period of time. It is impossible to quickly reproduce these changes on a fresh installation.

Another straightforward way of performing application migration is to simply copy an existing application in its entirety to virtual machines (VMs) in a cloud. The challenge here is to reconfigure the application such that it will still work in the new environment. As it turns out, handling network changes is a big part of this challenge. For example, one component may store the IP address of another component in its configuration file or database (that might even be encrypted). Without having intimate knowledge of the application, it is hard to know where the parameter is located.[2] Another example is the common practice of binding a software license to a specific IP address of the host machine. If the IP address later changes (e.g., after migration), the software will cease functioning, even with a valid license.

1.2 Contributions

The purpose of this paper is to present to the research community the challenges of migrating complex applications into cloud, and hopefully spark more research work in this area, which we, a group of cloud insiders, believe to be critically important in practice. This problem is particularly relevant to the network community, because changes in the network environment are often the culprit that breaks migrated applications.

We propose a methodology for defining the migration solution space, which helps enumerate a large number of potential solutions and discuss the pros and cons of each solution in a systematic way. We break down the migration problem into three tasks: OS migration, management tool migration, and application migration. A complete migration solution can be obtained by combining different methods for different migration tasks. In particular, we identify a rich set of solutions that potentially can run unmodified legacy applications by provid-

[1]It was reported that migrating the relatively simple Java Pet-Store benchmark to a cloud took more than 22 and 36 hours for preparation and migration, respectively [6]. From our own experience in a previous project, it took about a month to set up a solution that involved half a dozen different products.

[2]A concrete example is that MySQL may grant remote access for a database to a specific IP address. Access is denied if the IP address changes.

ing them with an illusion of an execution environment identical to the legacy environment.

In addition to defining the solution space, we propose two novel solutions, *AppCloak* and *PseudoApp*. AppCloak is an OS-level container that can run an unmodified legacy application by providing a disguised execution environment. AppCloak is radically simpler than traditional OS containers, because it is not concerned with security or performance isolation, which is the major source of complexity in the traditional containers. On the other hand, AppCloak goes beyond traditional containers — it provides migrated applications an illusion that their network environment is unchanged.

Another key observation is that the migration effort is often dominated by the pre-migration assessment rather than the actual migration. Tran et al. [6] reports that selecting proper cloud providers and server types requires significant effort during preparation for migration. Suppose an organization wants to assess which cloud out of three candidate clouds is the best fit (e.g., in terms of performance and/or cost) for each of its n applications, it has to perform a total of $3n$ migrations in order to complete the assessment. Suppose 50% of the applications are eventually considered not a fit for any cloud (i.e., only $n/2$ of the applications will be migrated), the number of pre-migration assessments is $3n$ — a 500% overhead!

Our solution, *PseudoApp*, addresses this problem by providing a quick way of assessing an application's performance (e.g., throughput and response time) and cost (e.g., due to VM size and disk/network traffic) in a potential target cloud, without migrating the actual application. It works by 1) building a simple pseudo-application that closely resembles the real application's behavior in resource consumption (CPU, disk, and network), and then 2) benchmarking the pseudo-application in the target cloud.

2. BACKGROUND

This section provides background on cloud and migration in order to set the stage for the discussion of migration solutions.

2.1 Cost Saving in Managed Cloud

We consider two types of IaaS clouds. For an *unmanaged cloud* such as Amazon EC2 [1], once a VM is provisioned, the user is fully responsible for managing everything running inside the VM. For a *managed cloud* such as IBM SmartCloud Enterprise+ (SCE+) [3], the cloud service provider charges a premium for managing software running in the VM. SCE+ provides a long list of managed services, including patching the guest OS, taking incremental backup of the guest file system, etc. These services are often enabled in a traditional IT environment through system administrators' labor-intensive manual operations, while are provided by SCE+ through full automation.

The cost of running an enterprise application is often dominated by the cost of providing the managed services rather than the cost of getting a basic physical or virtual machine. Moving the application into a managed cloud not only saves the infrastructure cost due to the economy of scale, but also replaces expensive labors with automation in providing the managed services.

2.2 Migration Tasks

A complete migration process involves three related but different tasks.

OS migration: The guest OS and device drivers used in a cloud are unlikely to be an exact match of those in the legacy environment. For example, the cloud may use a para-virtualized kernel. OS migration also covers basic system reconfigurations related to hostname, IP address, DNS server, etc.

Management tool migration: This is specific to a managed cloud, which installs many management tools in a VM to enable automation, e.g., Tivoli Endpoint Manager for patching. During migration, existing tools need to be replaced by the cloud's tools.

Application migration: An application usually has configuration parameters dependent on the legacy environment, e.g., IP address of a database server. These parameters need to be enumerated and modified before the application can run in the cloud.

2.3 Evaluation Metrics

We describe some important metrics that will be used to evaluate the migration solutions.

Ease of deployment: We prefer off-the-shelf software over custom implementation. If custom code is necessary, we prefer user-level code over kernel-level code.

Migration labor: We prefer solutions that need minimal human labor.

Post-migration maintenance: It is undesirable if a migration method packages the application in a stiff way that makes future application upgrade difficult.

Post-migration application performance: Some methods such as nested virtualization incur runtime performance overhead, which are less desirable.

Generality: Some methods may be simple but cannot handle many legacy applications.

Cloud provider support: If a method requires changes to hypervisor or network infrastructure, it is less attractive to application users compared with VM-level solutions, as they need to pay for the extra support even if the cloud provider offers the support.

3. MIGRATION SOLUTION SPACE

Table 1 shows the migration solution space. Each row represents a common method for OS migration and management tool migration. OS and management tools are

Application Migration / OS/Tool Migration	Reinstall app	Reconfig app	Wrap app	BlueCoat/Puppet
Create a new VM from the cloud's existing image catalog	✓	✓	See Table 2	✓(bc/Puppet)
Clone the old VM, adjust OS/tools in the image, and import it into cloud	N/A	✓(SCE+ Rapid Migr.)	See Table 2	N/A

Table 1: Migration solution space. Each migration solution is a combination of one method for OS and management tool migration and another method for application migration.

handled together because they jointly form the runtime environment for applications. Each column represents a method for application migration. A complete migration solution combines a method for application migration with a method for OS/tool migration.

3.1 OS/Tool Migration

There are two methods for OS/tool migration.

Create a new VM. This method simply provisions a new VM from the cloud's existing image catalog. Since the image is provided by the cloud, it already has the OS and management tools properly configured. However, the new VM's kernel, software packages, and library versions may differ from those in the legacy environment. The challenge then is to run the application properly in the new VM and put the application under the control of the installed tools.

Adjust the old VM. This method clones the old VM from the legacy environment, adjusts the image, imports it into the cloud, and finally provisions a VM from the image. The adjustments include changing the kernel, installing the cloud tools, etc. This method is difficult but doable, because OS/tool adjustments are a finite set, do not grow with the number of applications, and hence potentially can be automated. The advantage of this method is that, unlike a new VM, the adjusted VM already contains the application binary and configuration, which makes application migration easier.

3.2 Application Migration

We identify four methods for application migration.

Reinstall application. This method may be the most straightforward but time consuming approach.

Reconfigure application. This method copies the already installed application with its full configuration to VMs in the cloud. It then changes the application's configuration to make it work in the cloud, e.g., pointing one component to the new IP address of another component. The main challenge is that the way of reconfiguration is application specific and usually involves a lot of undocumented parameters.

Wrap application. This method copies the already installed application with its full configuration to VMs in the cloud. Rather than changing the application's configuration, it runs the application in a disguised execution environment, e.g., by performing network address translation (NAT) to hide the change of IP address.

PseudoApp. This method provides a quick and reliable way of estimating the application's performance in a target cloud. See Section 3.3.4.

3.3 Migration Solutions

Each cell in Table 1 represents a full migration solution. The solutions in the "*Wrap app*" column are explained separately in Section 4 and Table 2, because they have many variants.

3.3.1 S1: Complete Reinstallation

This solution provisions a new VM from the cloud's existing image catalog and then installs the application starting from scratch. This may be the most straightforward but time consuming approach.

3.3.2 S2: Get a New VM & Reconfigure App

This solution 1) provisions a new VM from the cloud's existing image catalog, 2) copies the already installed application with its full configuration from the legacy environment to the new VM, and 3) changes the application configuration so that it works in the cloud. It may start the application after `chroot` so that the application perceives the illusion of a file system identical to that in the legacy environment.

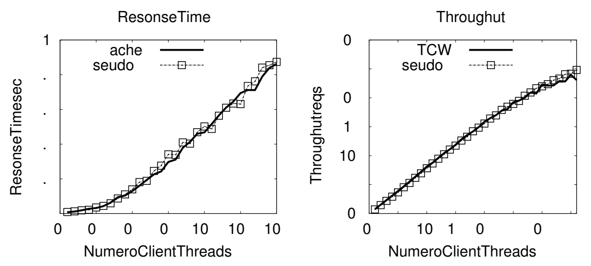
3.3.3 S3: Import the Old VM & Reconfigure App

This solution 1) adjusts the old VM and imports it into the cloud, and 2) changes the application configuration so that it works. There is no need to do `chroot` because the imported VM has the application binary and configuration at the original place.

3.3.4 S4: PseudoApp

This solution does not perform the actual application migration. It is a tool for a quick and reliable estimation of an application's performance in a target cloud. Then a user can quickly choose the right VM size, estimate the cost, and decide whether a real migration is worthwhile.

A PseudoApp resembles a real application's behavior of resource consumption at the thread level. For each component (i.e., process) in the real application, the PseudoApp has a corresponding pseudo component. Consider a multi-tiered web application. Processing a web



(a) Apache vs. PseudoApp (b) TPC-W vs. PseudoApp

Figure 1: PseudoApp's performance prediction.

request involves a sequence of nested invocations across multiple distributed components. A vertex in the sequence represents a processing step on a component, which performs certain CPU computation and/or disk I/O. An edge in the sequence represents one component calling another component through network.

When the same web request is submitted to both the real application and the PseudoApp, they produce identical invocation sequences, with the same order of traveling through different components. Moreover, within a processing step on a component, they execute the same order and the same amount of interleaved CPU computation, disk I/O, and network communication. The only difference is that, the real application does useful work, whereas the PseudoApp performs CPU spinning, writes to files with meaningless data, and sends network messages with meaningless payloads.

We have adopted the principles of vPath [5] to profile a distributed application and build the corresponding PseudoApp. vPath can track the precise invocation sequence of events relevant to resource consumption. We have implemented it to trace disk I/O and CPU service time, and enhanced it to handle event-driven programs. Due to the space limit, technical details are omitted and we present the preliminary evaluations of PseudoApp in Figure 1. We use the profiling data obtained under very light workloads to construct the PseudoApp for Apache and TPC-W (a multi-tiered ecommerce benchmark). The figure shows that PseudoApp is able to accurately predict the performance under high workloads, even if it never sees that workload during profiling.

4. WRAP-APPLICATION SOLUTIONS

This section presents migration solutions that belong to the "*wrap app*" column of Table 1. These solutions copy the application with its full configuration to VMs in the cloud, and then run the application in a disguised execution environment (e.g., performing `chroot` and NAT) so that the application functions properly without any configuration change.

4.1 Requirements on Disguised Environment

Below are a number of requirements for supporting a wide variety of unmodified applications in a new runtime environment.

FR1) The application sees the same file system layout as that in the legacy environment.

FR2) The application sees the same hostname as that in the legacy environment.

FR3) The application sees the same IP addresses as those in the legacy environment.

FR4) When the application performs a DNS lookup for an old hostname in the legacy environment, it gets the old IP address associated with that hostname.

FR5) Distributed components of an application see the same communication endpoints as those in the legacy environment. Suppose an application has two components X and Y. The VM running the component X uses old-IP-X in the legacy environment, but uses new-IP-X in the cloud. Similarly, component Y has old-IP-Y and new-IP-Y. In the cloud, when component X invokes `sendto(old-IP-Y)`, the message is delivered to component Y, even if Y's real IP now is new-IP-Y. Moreover, component Y sees the message coming from old-IP-X, even if X's real IP now is new-IP-X.

FR6) The application can initiate connections to and accept connections from servers outside the cloud. In the example above, a server on the Internet can connect to component X using new-IP-X. Component X can also directly connect to any server on the Internet.

We classify these requirements into two categories: 1) disguise network communication endpoints (i.e., *FR5* and *FR6*), and 2) disguise file system and everything else (i.e., *FR1–FR4*). Each row of Table 2 represents one method for handling the first category of requirements. Each column of Table 2 represents one method for handling the second category of requirements. A complete migration solution combines a method from a row with a method from a column.

4.2 Disguise File System etc.

chroot. This method 1) provisions a new VM from the cloud's existing image catalog, 2) copies the entire file system of the old VM in the legacy environment to a directory in the new VM (e.g., `/oldroot`), and 3) runs `chroot /oldroot` before starting the application. The application treats `/oldroot` as its root file system, which is identical to what it sees in the legacy environment. This simple method does not meet requirements *FR2–FR4*, but it is sufficient to make certain applications work, assuming they have only file system dependency and no other dependencies.

OS Container. To deliver a more faithfully disguised

Disguise File Sys. etc. / Disguise Net. Endpoints	Create New VM				Adjust Old VM (SCE+ Rapid Migr.)	
	chroot	OS Container	Misc syscall manipulation	Nested VM	OS Container	Misc syscall manipulation
Network Address Translation (NAT)	✓	✓	N/A	✓	✓	N/A
Virtual Private Network (VPN)	✓	✓	N/A	✓	✓	N/A
Network syscall manipulation	✓BCC	✓BCC	✓BCC	N/A	✓BCC	✓BCC:BlueCoat/Camouflage
Cloud provider enabled VPC/SDN	✓	✓	N/A	✓	✓	N/A

Table 2: Wrap-application solutions. Each checkmark ✓represents a valid solution, by combining a method from a row with a method from a column.

environment, this method creates in the VM an OS-level container (e.g., LXC [3]), configured with the old hostname (*FR2*), the old IP addresses (*FR3*), and `/oldroot` as the root file (*FR1*). By manipulating `/oldroot/etc/hosts`, it can return proper DNS lookup results (*FR4*).

Misc syscall manipulation. A traditional OS container is ill-suited for application migration. Its complexity mainly comes from enforcing security and performance isolation, which is not needed at all in application migration, because each VM runs only one container. A dramatically simpler approach is to just intercept and manipulate a few syscalls that are of interest to application migration, e.g., `uname()` for hostname (*FR2*), and `ioctl(SIOCGIFFLAGS)` for IP addresses (*FR3*). It can use `chroot` to provide a disguised file system (*FR1*), and manipulate `/oldroot/etc/hosts` to return proper DNS lookup results (*FR4*).

Nested VM. Running an OS container in a VM is one form of nested virtualization. Nested VM [2] goes even one step further. This method 1) provisions a new VM from the cloud's existing image catalog, which has the cloud required kernel and management tools, and 2) runs the old VM from the legacy environment as a nested guest VM inside the cloud VM. Users have the control over the nested hypervisor so that they can have VM management capabilities. On the other hand, it also has several major drawbacks: 1) highest runtime performance overhead among the solutions, 2) hard to deploy since no off-the-shelf implementation is available, and 3) increased post-migration maintenance cost, e.g., patching both the cloud VM and the nested VM.

Adjust the old VM. This is the method described in Section 3.1. The adjusted VM's file system is almost identical to that of the original VM, and hence there is no need to perform `chroot` before starting the application. Requirements *FR2–FR4* can be met using either the "OS Container" method or the "Misc syscall manipulation" method described above.

4.3 Disguise Network Comm. Endpoints

The network communication discussed in Table 2 can be disguised with or without the assistance of a cloud provider. The first three methods listed below do not require the assistance from a cloud provider.

Network address translation (NAT). To support *FR5*, for communication between two components of a distributed application, it needs to perform both source NAT (SNAT) and destination NAT (DNAT) on both endpoints (VMs running the application). Linux `iptables` supports both SNAT and DNAT, but it only allows SNAT for forwarded packets and disallows SNAT for packets that originated from an endpoint itself. *FR5*, however, requires the latter. It can be supported by adding a custom NetFilter kernel module like that in Zap [4] or hooking up NAT with an OpenFlow-like method which performs directory lookup for appropriate address translation.

Virtual Private Network (VPN). Instead of doing address translation, this method runs a VPN client in a VM, and forms a VPN among the components of a distributed application (encryption and authentication provided by VPN can be disabled for our purpose to reduce overhead). An off-the-shelf example is OpenVPN.[4] Typically, traffic between VMs is routed through the VPN server, which can impact performance. An alternative is to use dynamic multipoint VPN (DMVPN) so that endpoints can communicate directly.[5]

Network syscall manipulation. This method works at the syscall level to provide the illusion of a different IP address. Unlike NAT, it never makes any change to IP packets. See Section 4.4 for details.

Cloud provider enabled virtual private cloud (VPC). This method differs from the VPN method above in that, the VPC is implemented at the hypervisor level or network infrastructure level, rather than at the VM level. One example is Amazon VPC.[6] Though the internals of Amazon VPC are not published, there are many ways of implementing VPC. A VPC can use a VLAN to connect a customer's VMs and use a gateway to connect to the outside world. A VPC can also be implemented using software defined networking (SDN) in conjunction with

[3]http://lxc.sourceforge.net/

[4]http://openvpn.net/

[5]https://en.wikipedia.org/wiki/Dynamic_Multipoint_Virtual_Private_Network

[6]http://aws.amazon.com/vpc/

VLANs, VxLANS, GREs, mGREs, VPNs, or DMVPNs.

4.4 AppCloak

Due to the space limitation, we do not present the details of every solution in Table 2. We instead focus on our solution AppCloak, an OS-level container where legacy applications run without any modification. It intercepts a small set of syscalls and manipulates parameters and/or return values to meet the requirements of *FR2*, *FR3*, *FR5*, and *FR6*. Syscall interception can be implemented through kernel modification, `ptrace`, or intercepting the corresponding `libc` wrapper functions via `LD_PRELOAD`. We prefer `LD_PRELOAD` because it is easier to deploy than kernel modification and more efficient than `ptrace`.

To give the application the illusion of the old hostname, AppCloak intercepts `uname()` and returns the old hostname (*FR2*). AppCloak also intercepts `ioctl(SIOCGIFFLAGS)` and returns the old IP addresses (*FR3*).

AppCloak performs NAT at the syscall level rather than at the network layer. Suppose an application has two components X and Y, using old-IP-X and old-IP-Y before the migration, and using new-IP-X and new-IP-Y after the migration. Because old-IP-Y is stored in X's configuration file, X still attempts to invoke `connect(old-IP-Y)` to establish a connection with Y. AppCloak intercepts this and asks the OS to perform `connect(new-IP-Y)` instead.

When X invokes `getsockname()`, AppCloak returns old-IP-X. When X invokes `getpeername()`, AppCloak returns old-IP-Y. Similarly, when Y invokes `getsockname()`, AppCloak returns old-IP-Y. When Y invokes `getpeername()`, AppCloak returns old-IP-X. Therefore, X and Y perceive the same communication endpoints as those in the legacy environment.

AppCloak also does address translation for `accept()`, `bind()`, `sendto()`, `recvfrom()`, etc. However, when X `connect()` or `sendto()` a real IP address that should not be translated, e.g., an IP address outside the cloud, AppCloak just passes it through.

AppCloak has multiple advantages. First, it is easy to deploy, as it is a light-weight, user-level solution. Second, it is efficient. It does not inspect network packets, and does not even intercept the most frequently used syscalls for network communication, including `read()`, `write()`, `send()`, and `recv()`. Finally, migration with AppCloak is easy. It does not require time consuming reinstallation or reconfiguration of the application.

5. RELATED WORK

Zap [4] is the closest to AppCloak. It is an OS container that supports live process migration. It relies on a kernel module to perform NAT and migrate the kernel state. It assumes that the migration source and destination have similar environments, e.g., nearly identical kernel versions and a shared network file system for hosting application data. AppCloak's user-level implementation is much simpler than Zap, because AppCloak does not target live migration. AppCloak supports migration between more heterogeneous environments, e.g., para-virtualized vs. vanilla kernels.

Darwin [7] is a tool capable of migrating applications across different platforms. It takes the reinstallation approach but uses the configuration discovered in the old environment to guide the installation and configuration in the new environment. It can only migrate software for which it has built-in knowledge.

6. CONCLUSION AND FUTURE WORK

We presented the problem of application migration into cloud, and developed a taxonomy of migration techniques based on which software components to migrate and how to handle network and file system encapsulations. We found that these solutions have different challenges and trade-offs. In addition, we proposed *AppCloak*, a technique for running a legacy application in a cloud without modifications, and *PseudoApp*, a technique that assesses an application's performance in a cloud without installing the real application.

For future work, we plan to evaluate and compare the different migration solutions on a wide variety of real applications. Clearly, no one solution has the absolute advantage over all the others. It is important to identify a sorted list of environment factors that are "sensitive" to applications, i.e., their changes are likely to break applications. This helps evaluate whether a migration solution addresses those sensitive factors.

More experiments are needed to evaluate each solution's impact on the post-migration maintenance of the VM and application. For example, it is unclear whether "chroot+AppCloak" would cause issues for application upgrade, system library upgrade, or distro upgrade.

The complexity can be avoided if the cloud provider builds full migration support into the cloud. The design of such a cloud is still an open question. Ideally, the cloud should 1) require no changes to an imported VM except installing cloud management tools, 2) support network communication for an imported VM even if it uses its old IP address, and 3) enable the cloud management tools to work properly, even if they function under the VM's old IP address, hostname, and other legacy states.

7. REFERENCES

[1] Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`.

[2] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The turtles project: design and implementation of nested virtualization. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[3] IBM SmartCloud Enterprise Plus. `http://www.ibm.com/services/us/en/managed-cloud-hosting/`.

[4] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The design and implementation of zap: a system for migrating computing environments. *SIGOPS Oper. Syst.*, 36(SI):361–376, Dec. 2002.

[5] B. C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang. vpath: precise discovery of request processing paths from black-box observations of thread and network activities. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 19–19, Berkeley, CA, USA, 2009.

[6] V. Tran, J. Keung, A. Liu, and A. Fekete. Application migration to cloud: a taxonomy of critical factors. In *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing*, SECLOUD '11, New York, NY, USA, 2011. ACM.

[7] C. Ward, N. Aravamudan, K. Bhattacharya, K. Cheng, R. Filepp, R. D. Kearney, B. P. andk Larisa Shwartz, and C. C. Young. Workload migration into clouds challenges, experiences, opportunities. In *IEEE CLOUD*, 2010.