國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

以使用者為導向雲端部署服務之選擇及成本預估

CloudGuide: a Tool to Help Users Estimate Cloud Deployment Cost for Legacy Network Applications

劉秀慧

Liew Siew Huei

指導教授：蘇雅韻博士

Advisor: . Ya-Yunn Su, Ph.D.

中華民國 101 年 8 月

August, 2012

致謝

在台灣求學的六年時光，不知不覺中就匆匆度過了！現在，台灣儼然是我第二個家，我在中央大學及台灣大學度過了許多寶貴的時光，若非它們，也不會有現在的我。

能夠順利完成碩士的學位，首先要感謝我的指導教授蘇雅韻教授。謝謝您包容我的無知，給了我極大的探索與犯錯的空間，因為您的悉心指導，讓我兩年內成長了不少，也了解了研究的苦與樂。感謝周承復教授，在我研究的過程中給了我信心與鼓勵，及許多無論是研究上或生涯上的幫助。感謝我的口試委員孫雅麗教授及劉邦鋒教授對我論文的指導，令這份論文的內容能夠更加完善。此外，我也要感謝我在中央大學的大學專題指導教授顏嵩銘教授及導師張貴雲教授，若不是您們的鼓勵，我將不會進入研究所就讀。

這兩年來交了不少新朋友，我想特別感謝實驗室的東耿、哈克、全毅、振新、峻毅、冠宇、立凱、嘉偉及易展，在實驗室與我共度了許多日常，無論開心的日子或不開心的日子，都有你們的參與。感謝我的室友一琳、好友儀雲，及大學好友逸文、丕獅、享倫、依琴，謝謝你們與我分享了不同的生活經驗，彼此鼓勵。還有從小與我一起長大的摯友媛君、馨怡及婉君，謝謝你們一直無條件地支持我，傾聽我的抱怨與苦水。感謝一樣與我身在異鄉的阿姨、姨丈及一家人，謝謝你們平日諸多的照顧，令我即使遠在他鄉，依然不寂寞。

最後，謹以此篇論文獻給我的家人：我的爺爺、父母、姐姐、弟弟，及去年去了天堂的奶奶，你們是我心中永遠的支柱，希望不久後的將來，換我作你們的支柱。

Abstract

With cloud business gowning, many companies are joining the market to be a cloud service provider. Most providers offer similar services with slightly different charging models, and very little known performance comparison between them is published. This leaves cloud service users with the puzzle of guessing what cost they will need to pay to run their legacy application in a cloud environment. CloudGuide is a tool suite that provides user an estimated cost of running a legacy application with QoS guarantee on different cloud providers. CloudGuide predicts the cloud computing resources required by a targeted application based on queuing model, meanwhile estimates monetary deployment cost for the application. CloudGuide also allows the user to explore cloud configuration can guarantee different levels of QoS. In our evaluation, we conducted experiments of a multi-tiered network application, RUBiS, and showed that CloudGuide can choose cloud configuration that guarantee QoS and provide cost estimation.

KEYWORDS

Cloud Computing, Capacity Planning, Public-Cloud Providers, Web Application, Pricing, Performance, Comparison.

中文摘要

隨著硬體及網路的進步，1966 年提出的運算資源即是公用資源的概念（Computing as a Utility），終於得以實現，並在近年內取得了商業上的矚目，它即是現今的雲端運算。雲端運算提供了一個嶄新的 IT 服務模型——應用程式及服務供應商不需要自行購置數據中心，轉而向雲端供應商依需求租賃運算資源。然而，現今雲端供應商眾多，各家的硬體資源及計價方式都不盡相同，在缺乏標準化及資訊不公開的情況下，雲端使用者難以比較不同的雲端服務的效能及價格，並確保雲端的移植確實帶來成本的節約。在此等前提下，對一個欲將自身 Web 應用程式移植至雲端平台的使用者來說，如何依程式的特性，來選擇適當的雲端平台，是一件困難的事。本研究致力於提供一套成本及效能導向的工具組，依程式的特性及使用者的效能品質需求，來選擇適當的雲端平台及預測所需要的部署成本。

關鍵字

雲端運算，容量規劃，公共雲端供應商，Web 應用程式，計價模式，效能評量及比較。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing has been predicted to be a disruptive technology that impacts how IT services are delivered. Application and service providers no longer need up-front commitment on building their own IT infrastructure; the elasticity of cloud resource pool and pay-as-you-go pricing model promises capital cost reduction and faster time to market [5].

As public cloud computing services gain popularity, many companies are joining this increasingly competitive and profitable market to become cloud providers. There are well-known international providers whose datacenters spread across multiple countries, for example Amazon Web Service [1] and RackSpace [4], as well as domestic providers targeted at local business, for example Hicloud [2] and NTU Cloud [3].

While many startups have already chosen cloud platforms to deliver their new services, well-established enterprises also consider migrating their legacy applications to the cloud. Users are often interested in the common capacity planning questions: "How many cloud resources from which provider should be provisioned? How cost effective they are? What are the estimated deployment cost?

While answering such questions, users might encounter several challenges:

- **Diverse cloud provider, service and pricing model** Due to lack of standardization, every provider presents different pricing models and shows diverse cloud performances, therefore, users find it difficult to establish a baseline to compare providers. The diversity of cloud offerings is increasing as new providers continue to emerge and old providers introduce new services.

- **Mispredictions** Users are interested in capacity planning solution for a particular application. To yield correct configurations for hosting such application, users need to predict the performance characteristic on different cloud platforms. This can be challenging due to several reasons. First, different types of application utilize resources (i.e. memory, CPU, disk etc.) differently. Users need to find an adequate prediction methodology for that application. Second, the available information about these characteristic can deviate from actual cloud performance [31]. Also, the underlying virtualization technique introduces interference between tenants, which further leads to performance over- or under-estimated.

- **Limited time and monetary budget** Under the circumstances that users have limited time and monetary budget, they can only explore a subset of cloud configurations from today's market. The speed and the monetary cost of finding a valid configuration effect the number of possible configurations that users can explore.

Some pragmatic solutions have been proposed and used in practice. Cloud providers offer a price calculator [1, 2] that users input the number and type of resources, and the calculator outputs an estimated cost. This approach requires users to guess the amount of cloud resources they need to sustain their application performance requirements and does not offer cross cloud provider comparison. Cloud benchmark tools [18] provide performance and cost benchmarks which can be informative but the users still need to translate the benchmark results into their own application-specific requirements. Finally, work on capacity planning in the cloud [22] offers solution in selecting cloud instances but with limited policy options and omitting I/O cost.

In this paper, we provide CloudGuide, a tool suite that helps users on capacity planning and deployment cost estimation for legacy, replicable Internet application with QoS guarantee. We help users to answer the questions:(1) among multiple resource configurations available for a targeted application, which one to choose, (2) what is the expected monetary cloud deployment cost for hosting such application. CloudGuide can also be leveraged to explore "what-if" scenario, such as capacity planning for reduced response time or increased throughput.

In particular, we make the following contributions:

Comparison across diverse cloud providers Current cloud providers offer virtual IT resources with slightly different semantics, hardware specifications, and pricing models. No standard is established among cloud resources meaning that users need to deploy their application on each one to understand the delivered performance and incurred cost of each provider. CloudGuide provides users with insight of what the predicted performance and estimated cost if the application were to migrate to a specific provider. CloudGuide achieves this by using generic benchmark results and CPU events combined with queuing model to predict application performance.

Providing insight for what-ifs exploration One advantage cloud computing provides is an easy way to cheaply test their application under different hardware models. However, since users only have machines in their own local cluster as references, it is not easy to know what the performance would be without deployment. CloudGuide is designed so that a user can run the application resource profiler once and explore the what-if scenarios where their application run on a different hardware model. CloudGuide provides a cost-calculation engine that allows users to see predicted performance and cost in a cloud environment compared to their local cluster. This flexibility allows the users to explore their options before start running their applications.

Dynamic provisioning when workload changes Since cloud computing promises the flexibility to scale-up dynamically, one important mechanism is to identify when and how to provision so that service level objective (SLO) is maintained when workload increases. In this work, we define application SLO as response time or throughput. CloudGuide models Web applications using queuing model to predict the number of instances required to meet SLO when workload increases and updates the cost so that the user can know in advance given a workload forecast which kind of operation cost is expected.

We begin with a problem statement in Chapter 2 and describe how CloudGuide predicts the performance of a migrated application in Chapter 3. Chapter 4 presents CloudGuide implementation, followed by Chapter 5 showing the effectiveness of CloudGuide. Chapter 6 discusses related work, and finally, Chapter 7 concludes the paper.

Chapter 2

Problem Statement

Consider a user with a legacy application that she would like to know the estimated cost and performance were the application migrated to the cloud. The user is faced with N providers with various computing instance types and storage options; each provider has a pricing scheme that differs slightly from each other. The specification for each computing instance may be arbitrary, with each provider naming their instance in different terms. For example, Amazon uses the term EC2 computing unit [1] while HiCloud uses HiCloud computing unit [2] to specify the computing power their instance. It is not clear given a legacy application, what the expected performance on an instance type would be.

On top of different instance type, the pricing of instances may not be linear within a cloud provider. Sharma et al. [22] pointed out that pricing for both EC2 cloud and NewServer (NS) cloud platform is convex, meaning the cost per-core increases sub-linearly as the number of cores increases.

The goal of CloudGuide is to provide a user with suggestion based on predicted price and performance according to her desired policy, such as minimized cost or maximized performance. CloudGuide can be invoked in initial planning or be executed whenever observed performance deviates from a priori goal.

Table 2.1: Classification of cost to host an application in the cloud

Cost type	Resource type	Calculation
Computing resources	Compute instance	Number of instances * Unit price * time
	Network	Data rate * time * Unit price
	Storage -- storage	Total data size * Unit price
	Storage -- transaction	Transaction rate * time * Per transaction cost
Services	Content distribution network	Commit rate + overage charge
	Load balancer	Fixed rate + usage-based for data access
	Resource monitoring	Fixed rate + usage-based data access
Total		Cost for computing resources + services

2.1 Cost calculation

The cost of running an application at a stable state in the cloud can be roughly divided into two parts: (1) *computing resources cost* includes cost paid to finish the actual application work. This cost depends on the performance requirements for the application and (2) *service cost*, additional service to improve the application execution and is often optional. Table 2.1 shows an example of the different cost type we include when estimating deployment cost for applications in the cloud. We focus on computing resource cost because service cost is optional and often is incurred to improve managing the application.

Computing resource cost is expressed as: $Cost_{running} = Cost_{Compute\ instances} + Cost_{Network} + Cost_{Disk}$. When calculating $Cost_{Computing\ instances}$, the number and type of instances required to host an application is subject to performance requirements based on the SLO from users. Similarly, the type of storage option also affects application performance and storage cost. Both network and storage cost can be calculated using a linear formula. Note that each provider charge in slightly different scheme, but we find this formula sufficient to capture across different providers.

2.2 Performance requirements

CloudGuide needs to predict application performance in the cloud to ensure performance requirements are met. Since we focus on web applications, we define performance as both *response time* within a time threshold, say two second, while supporting a given *throughput*. This prediction problem can be rephrased as ``How many resources should a user request from a provider so that the performance requirements can be met?"

Since cloud resources are often provided into three categories: computing resources (including memory), network, and storage. Estimating the amount of resources required is reduced three subproblems: determining the number of instances of a specific type, which type of storage and which type of network. In this work, we focus on the first subproblem. Determine the suitable type of storage is non-trivial and often requires application modification as demonstrated in work by Kossmann et. al [15], where they compare performance of running database under different architecture in cloud. In this work, we assume the most basic local storage option that usually comes default with a computing instance, such as elastic block storage (EBS) in Amazon and default local storage for HiCloud computing instances. Network cost can be considered as a linear term in current practice.

Consider a user plans to migrate a three-tier application, with each tier denoted $T_i, 1 \leq i \leq 3$, to the cloud. We do not consider replication on database tier since it is beyond the scope of this work. Consider across all cloud providers we observed, they in total offer K different types of instances, for example, Amazon EC2 small, medium, large, Hicloud small, medium, large. The application-specific cloud configuration can be denoted as a set

$$C = \{\sum_{j=1}^{k} n_{1j}, \sum_{j=1}^{k} n_{2j}, \sum_{j=1}^{k} n_{3j}\}, \tag{2.1}$$

where n_{ij}, $n_{ij} \geq 0$ denotes the number of instance type j hosting tier i. n_{ij} being zero means instance type j is not chosen to host tier i. Note that there can be a mix of instance types chosen, for example, three medium instance and one large instance from Amazon EC2.

2.3 Satisfying requirements based on policy

Given an SLO requirement of peak throughput λ_i for tier T_i, let p_j and μ_j denote that the price and the server capacity of instance type j. Here we define server capacity as to the application-level throughput a server can sustain. The problem can be formulated according to their policy.

2.3.1 Policy I: Minimize deployment cost while satisfying peak throughput

$$min \sum_{j=1}^{k} p_j n_{ij} + Cost_{network} + Cost_{storage} \tag{2.2}$$

such that

$$\sum_{j=1}^{k} \mu_j n_{ij} \geq \lambda_i \tag{2.3}$$

where n_{ij} denotes the number of instance type j hosting tier i. We formulate this optimization as an integer linear programming (ILP) to solve the n_{ij}. The solution yields a set of values (n_{11}, n_{12}, ..., n_{1K}) for tier 1 and (n_{21}, n_{22}, ...,n_{2K}) for tier 2. Recall that K represents the total number of available instance types from all cloud providers we observed. To disable a configuration composing of instance type across different cloud providers in a configuration (i.e. hybrid cloud deployment), we can limit K to instance types from a particular cloud provider one at a time (e.g., set k to be from instance types 0 to 3) , and run ILP iteratively until all cloud providers have been taken into consideration.

Note that meeting performance requirements usually can be met with two ways: *scale-out*, meaning deploying more servers to handle the workload, and *scale-up*, meaning using more powerful server to handle workloads. In CloudGuide, we combine both approaches by finding n_{ij} for *scale-out* and finding server with different capacity μ_j for *scale-up.*

2.3.2 Policy II: Maximize throughput given monetary budget M

$$max \sum_{j=1}^{k} \mu_j n_{ij} \tag{2.4}$$

such that

$$\sum_{j=1}^{k} p_j n_{ij} + Cost_{network} + Cost_{storage} \leq M \tag{2.5}$$

where M is the monetary budget for T_i. The total cost of the cloud configuration should not exceed this budget, while maximizing throughput.

Users can also explore what-if scenario by manipulating parameters such as SLO [1] or workload λ_i, as the example shown below

2.3.3 Policy Example: Improve throughput by half and keep minimum cost

$$min \sum_{j=1}^{k} p_j n_{ij} + Cost_{network} + Cost_{storage} \tag{2.6}$$

[1]Changing SLO requires server capacity estimation from Chapter 3 to be re-run. For example, if user wants to reduce average response time by 20%, the user needs to specify the new average response time to $d' = d * 80\%$, and re-calculate the server capacity.

such that

$$\sum_{j=1}^{k} \mu_j n_{ij} \geq \lambda_i * 150\% \tag{2.7}$$

To solve the above ILP problem for a policy, a perfect solution can be obtained using a solver, such as CPLEX and `lpsolve` solver. However, as the problem sizes grows, it takes quite long time to obtain a perfect solution. For policy I and II, we map the problem into a *min or max knapsack problem* and implement an approximate greedy algorithm which is worst-case bound of 2 [22]. We will describe how we find the parameters required to solve this problem in Chapter 4.

2.4 Design goals

We design CloudGuide following principles below:

Application-specific suggestion Our primary goal is to assist a customer to choose a cloud provider for its application by estimating the application performance and monetary deployment cost on various providers. Recent research shows that there is no clear winner from all aspect in the current market [17], therefore different types of applications do not always best-suited to be deployed on one single cloud provider. We believe incorporating applications characteristics into the suggestion procedure is more likely to yield meaningful results as it is more relevant to the targeted applications.

Unobtrusive and transparent We aim to provide a light-weight tool which can be transparently used in common production environment. In order to make application-specific suggestion, CloudGuide requires application log files and resource usage profile as input. We believe that application log files are easily obtainable by system administrator, and we adopt popular profiling tools, `sysstat` [26] and `Oprofile` [20], which are commonly accepted to be unobtrusive to collect resource usage and CPU event profile. Once fed with required input files, CloudGuide should provide suggestion results in a reasonable short period of time (e.g. within seconds or minutes). We adopt G/G/1 and G/G/m queuing models in our work to speed up the suggestion process, eliminating the needs of migrating applications to every candidate cloud providers.

Modularized design Modularity in design enables every component of CloudGuide substitutable. For example, a new technique that more precisely predicts application performance can transparently replace current method adopted by CloudGuide. Modifications can be easily done on CloudGuide to quickly adapt to cloud services changes.

Chapter 3

Meeting Performance Requirement

Meeting user performance requirements is essential for CloudGuide. As SLO is specified in terms of response time and throughput, CloudGuide needs to estimate the capacity each instance type can sustain for the targeted application. Estimating capacity accurately is crucial for CloudGuide to propose a cloud configuration that can provision sufficiently without wasting resources to the targeted application. Note that CloudGuide makes *conservative* estimation to ensure SLO are satisfied.

To capture the application performance on a unseen platform, one approach is to run the application in that platform, drive with test workload, and measure the maximum throughput. Previous work [22] took this empirical approach to determine server capacities. We argue that in cloud environment where providers offer multiple instance types, this approach can incur high cost because without knowledge on which size suits the application, one needs to try all. Moreover, this approach requires the extra porting effort to every candidate cloud provider.

We adopt the approach of estimating maximum throughput of each cloud instance type by modeling the server as queuing system. Previous work has shown that network applications can be modeled well as queuing systems [10, 25, 24, 29]. The results from queuing theory can be used to derive maximum throughput from request rate, service times, and the response time specified in user SLO. We took this approach because it requires parameters that can easily captured locally so it saves users the porting efforts and monetary cost.

Our idea is to start by using G/G/1 queuing system to model the server capacity on a known platform (i.e., local machine hosting the target application), then estimate the relationship between server capacity between local machine and each cloud instance from benchmark results. Even though both follow queueing model, there is a *speedup factor* between local server capacity and cloud server capacity. CloudGuide applies techniques to *cross-platform performance estimation* to derive such factor. The three steps taken in our approach to estimate server capacity can be summarized as followed:

1. Establish server capacity from local server using G/G/1 queuing model.
2. Calculate speedup factor for each cloud instance type.
3. Derive estimated server capacity for each cloud instance type by combing local server capacity and speedup factor using G/G/m queuing model.

3.1 Establishing base capacity

We applied the following formula from G/G/1 queuing system [14] to calculate the baseline server capacity:

$$\lambda_i \geq [s_i + \frac{\sigma_a^2 + \sigma_b^2}{2 * (d_i - s_i)}]^{-1} \tag{3.1}$$

where d_i is the mean response time for tier T_i, s_i is the mean service time of requests at tier T_i, σ_a^2 and σ_b^2 are the variance of inter-arrival time and variance of service time, respectively. Note that d_i is the response time requirement defined in SLO and rest of the parameters can be obtained from application logs. Here, d_i can be specified either as average response time or high percentile of response time distribution, for example, the end-to-end response time of 95% of the requests below 2 seconds. Finally, we derive λ_i as the lower bound of input request rate that can be served by a single server of tier T_i.

3.2 Determining capacity for different instance type

After establishing the server capacity baseline, we approximate server capacities of different instance types by modeling them as G/G/m queuing system. Our idea to model cloud server as G/G/m is because we assume that the application can be modeled as queuing system and the main difference between servers is their service time. First, we replace the service time to be *m times speedup* in equation 3.1 to derive capacity [14] as followed:

$$\lambda_i \geq [\frac{s_i}{m} + \frac{\sigma_a^2 + \frac{\sigma_b^2}{m} + (\frac{m-1}{m^2})s_i^2}{2 * (d_i - s_i)}]^{-1} \tag{3.2}$$

where we reuse all parameters $d_i, s_i, \sigma_a^2, \sigma_b^2$ from equation 3.1, except we need to find the speedup factor m. This is because we assume the request arrival rate follows similar distribution. Any workload prediction method can be applied here to improve the accuracy of request rate. To find m we need to estimate how the application service time might change on a unseen platform.

3.3 Finding speedup factor

We focus on finding speedup factor m for business logic tier because it is generally computing intensive that can benefit from instance type changes and it takes largest portion of overall response time. On the other hand, front-end web tier that serves small static files is very lightweight comparing to business logic.

Note that we are finding speedup factor compared to local machine instead of absolute average service time. This is a simplified version of cross-platform application performance prediction, and any technique for solving this problem can be applied here. One simple way to find speedup factor is to use the ratio of benchmark scores, e.g., using SPECjvm score. However, this approach requires finding a benchmark that behaves similarly to that of the targeted application.

Previous work on cross-platform application prediction adopted machine learning technique [16], modeling technique [13, 23, 24], or trace-and-replay [18]. However, not all approaches are applicable to cloud environment because of limited access to hardware level performance metrics. For instance, work that relies the program similarity on micro-architecture characteristics [13] is hard to apply in cloud environment because many characteristics are virtualized in cloud environment. CloudGuide adopts the performance model from [24] which predicts application-level performance for Internet services using platform parameters, including processor speed, cache and memory latencies. They note that instruction and memory-access latencies are key components of the processing time of individual requests for Internet service [24].

The average service time of tier T_i is expressed as:

$$s = \frac{I * (C + H_1C_1 + H_2C_2 + M_2C_{mem})}{number\ of\ requests} \tag{3.3}$$

where I denotes the aggregate number of instructions observed in T_i, C is the average service time per instruction (not including memory access delays), H_k is the percentage of hits in the L_k cache per instruction, M_k is the percentage of misses in the L_k cache per instruction, C_k is the typical time access to the L_k cache, and C_{mem} is the time access to main memory. We derive I, H_k, M_k from CPU events profile, and capture C, C_k, C_{mem} from memory benchmark results. Note that CPU events profiling only needs to be run

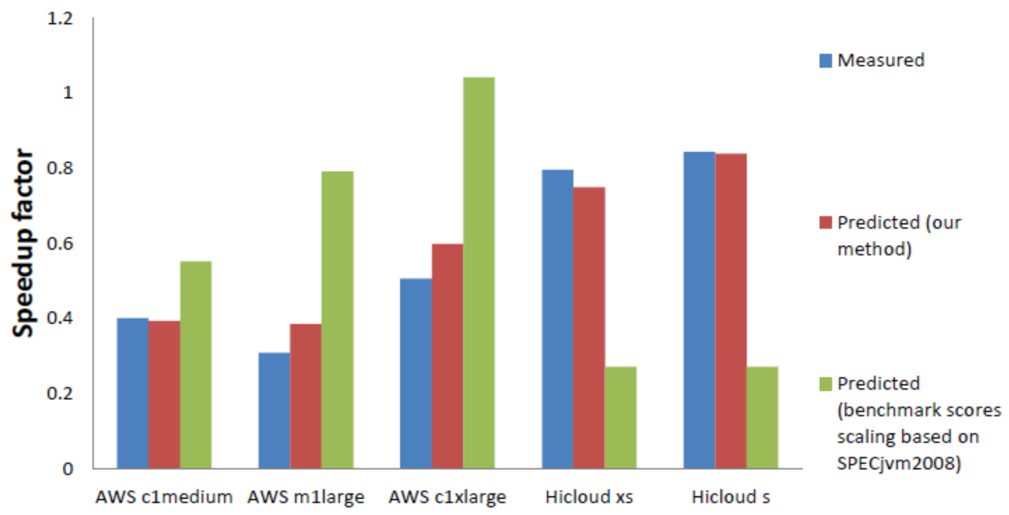


Figure 3.1: Speedup factor predicted by our method and by benchmark scores scaling for RUBiS. Workload includes read-only requests.

locally and memory benchmarks need to be executed once on each cloud instance type.

Speedup factor for a instance type j compared to local can be derived using the following formula, where both service times are derived from equation(3.3).

$$m_{ij} = \frac{s_{ij}}{s_{local}} \tag{3.4}$$

Figure 3.1 and figure 3.2 show the accuracy of predicting speedup factor for RUBiS using our method, compared to measured speedup factor from real deployment and speedup factor predicted by SPECjvm2008 benchmark scores. The speedup factor is the ratio of performance enhancement or degradation in terms of service time after an application is migrated from on-premise to corresponding cloud instance. Even though SPECjvm2008, a benchmark suite for measuring the performance of a Java Runtime Environment (JRE) has high similarity to application tier, prediction results of our method are better than benchmark scores scaling.

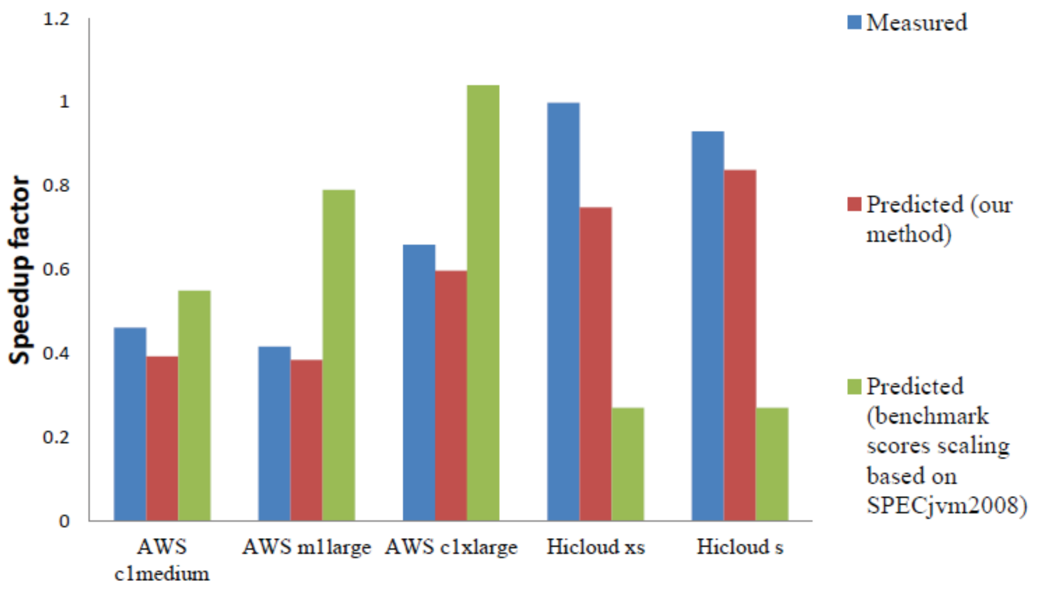


Figure 3.2: Speedup factor predicted by our method and by benchmark scores scaling for RUBiS. Workload is a bidding-mix including 80% read requests and 20% write requests.

Chapter 4

Implementation

4.1 Overview

We describe the implementation of each step CloudGuide follows as shown in Figure 4.1 in this chapter:

1. Conduct up-to-date cloud services and pricing schemes survey by using web crawler. Meanwhile, establish performance comparison baseline among cloud providers by benchmarking.

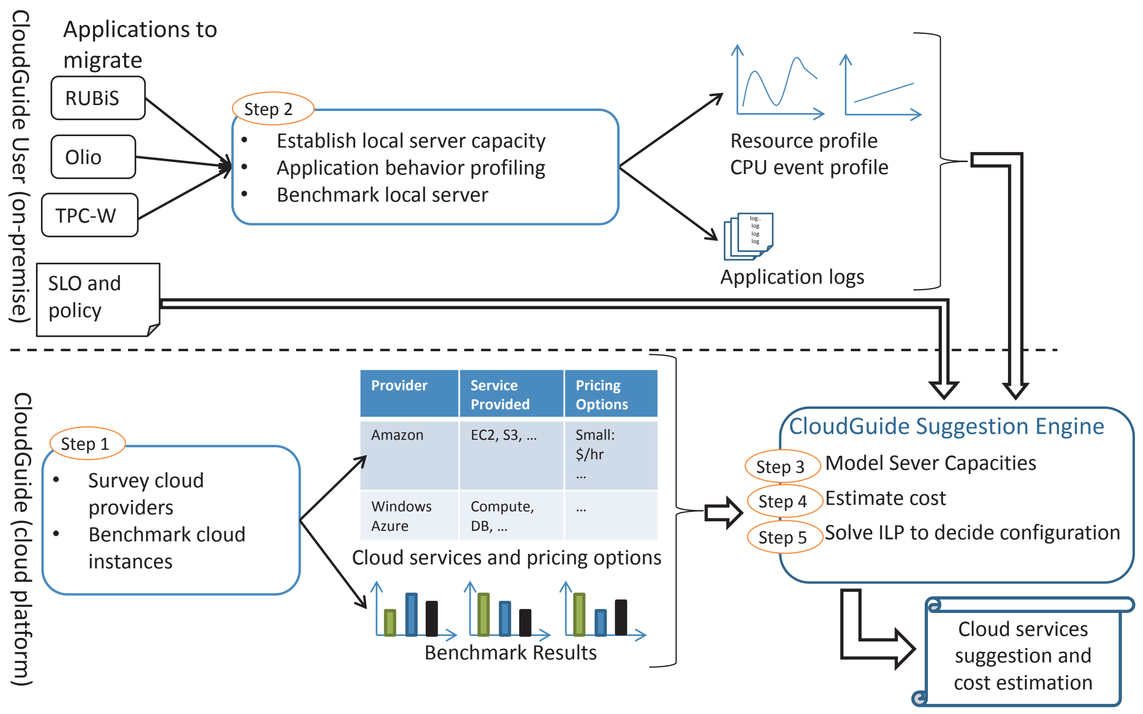
2. Establish local server capacity from application queuing behavior, profile application I/O usage, and conduct benchmark for comparison with cloud instances.

3. Model server capacity of different types of cloud instance for the targeted application by using queuing model and benchmark results.

4. Estimate monetary cloud deployment cost for that application, in accordance with the cloud configuration.

5. Determine optimal cloud configuration for that application, according to user-specified SLO and policy.

Note that output from the first two steps server as input to *suggestion engine*, which completes steps three through five. Users also need to specify two parameters for suggestion engine: SLO (i.e. application response time) and policy. If SLO is not explicitly


CloudGuide User (on-premise)
Applications to migrate
RUBiS
Olio
TPC-W
SLO and policy
Step 2
Establish local server capacity
Application behavior profiling
Benchmark local server
Resource profile
CPU event profile
Application logs
CloudGuide (cloud platform)
Step 1
Survey cloud providers
Benchmark cloud instances
Cloud services and pricing options
Benchmark Results
CloudGuide Suggestion Engine
Step 3
Model Sever Capacities
Step 4
Estimate cost
Step 5
Solve ILP to decide configuration
Cloud services suggestion and cost estimation

Provider	Service Provided	Pricing Options
Amazon	EC2, S3, …	Small: $/hr …
Windows Azure	Compute, DB, …	…

Figure 4.1: CloudGuide overview

specified, the default behavior of suggestion engine is to use the hardware specification of the local cluster as a reference and search across virtual hardware options from all cloud providers for configurations that offer similar performance to host the targeted application. Policy is what impacts suggestion engine directly on making choice among configurations. CloudGuide currently supports two polices: minimize deployment cost, and maximize throughput under given monetary budget. We plan to extend CloudGuide by adding more supported policies, for example minimum average response time. If users are interested in exploring what-if scenario, they can leverage suggestion engine by specifying different SLO for potential development preferences, such as reduce response time by 20% or increase throughput by half while keeping minimal cost.

Table 4.1: Cloud providers chosen

Provider	Type	Geo-location	Time-based pricing	Usage-based Pricing
HiCloud	IaaS	Taiwan	CPU (Daily)	Network
NTU Cloud	IaaS	Taiwan	Whole machine (Yearly)	
Amazon AWS	IaaS	US, Ireland, Tokyo, Singapore	Instance (Hourly)	Network and disk

4.2 Cloud Providers Pricing Schemes and Performance

First of all, we start with survey cloud providers' pricing scheme and establish performance comparison baseline among cloud providers. This process can be configured to either re-run periodically, such as after weeks or months, or triggered manually by users, such as, upon new service update or price change. The results obtained from this step are reusable for any CloudGuide user. Users do not need to participate in this step to use CloudGuide, but users can also conduct their own survey and benchmarks if necessary.

In the subsequent subsection, we explain the cloud providers we choose to observe, and how CloudGuide establish performance comparison baseline among them.

4.2.1 Chosen cloud providers

When choosing which cloud platform to move one's application to, we first need to decide which type of cloud service based on the common characterization of infrastructure-, platform- and software-as-a-service (i.e. IaaS, PaaS, and SaaS). For users developing new applications, they have the freedom to choose the appropriate level since they are starting from a clean-state design; however, others may not be so lucky. In our case, we target on migrating legacy network application to the cloud, so we choose cloud providers who provide IaaS service, which provide virtual machine abstraction that imposes least limitations.

We list four chosen cloud providers in Table 4.1 along with their information, including service types, geo-locations, and resource charging schemes. We choose cloud providers with the following principles: (1) providers that provide IaaS, (2) covering both domestic (e.g. HiCloud) and international cloud providers, (3) popular and representative cloud providers. From the table note that two common pricing schemes are used: (1) time-based, where user pays the provider with the amount of time elapsed to use the resource, and (2) usage-based, where user pays the provider based on the actual resource used. The time-based pricing scheme, which its time unit is usually in terms of hours or days, is similar to parking-fee payment, where usage-based pricing scheme is closer to how utilities, such as power and water, are charged.

4.2.2 Performance Comparison Baseline

The performance of different resources (e.g. CPU, memory latencies, disk, network bandwidth etc.) of a cloud instance impact how an application hosted on that instance will perform. To find a desirable configuration, we need to compare the cost and performance among cloud providers. While some of the cloud providers disclose performance information of their cloud services (e.g. Amazon Web Service Elastic Cloud Compute disclose computing power for each type of instance in terms of ECUs, where ECU is a custom-defined computing unit), those information might only cover partial of all resources. Moreover, lacking standardization made performance comparison among different cloud providers non-intuitive. Prior work [17, 15, 21, 30] provide solution to this problem by benchmarking cloud providers.

In CloudGuide, we use `lmbench` microbenchmark to establish performance comparison baseline between different cloud instance type. We measure the hardware capabilities

Performance Counter Event (mask)	Description
INST_RETIRED (0x1)	number of instructions retired
L1I (0x02)	number of instructions fetches that miss the L1 cache
L1I (0x03)	number of all instructions fetches at L1 cache
L2_RQSTS (0x20)	number of instructions fetches that miss the L2 cache
L2_RQSTS (0x30)	number of all instructions fetches at L2 cache

Table 4.2: Events used in CPU events profiling

(e.g. time to finish a certain amount of floating point operations, cache and memory access latency) of different instance types from all cloud providers we observed. Other cloud benchmark service [17, 7] can be used to substitute current benchmark method as well.

4.3 Estimating local and cloud server capacity

We aim to estimate server capacity of each instance type when it is used to host a targeted application. We implemented the cross-platform performance prediction technique to approximate the server capacity for each instance type as described in Chapter 3.

We first determine local server capacity using G/G/1 model, which states that server capacity can be derived using equation 3.1. We instrument application tier and analyze application logs to find the variance of inter-arrival time and variance of service time.

Using the cross-platform performance comparison method describe in Chapter 3, we can derive *speedup factor* between a cloud instance type and local. To derive speedup factor for each cloud instance type, we first measure the number of CPU instructions and hit/miss ratio using `oprofile` on local machine. Table 4.2 shows the corresponding CPU events we profiled. Then, we use `lmbench` benchmark to establish cache and memory access time on both local and cloud instances.

After finding local server capacity and speedup factor, we finally can find cloud server capacity using equation 3.2. We implement this calculation using `python` script.

4.4 Finding configuration and cost that satisfy policy

To solve the ILP problem for each policy described in Section 2.3, we map this problem into min- or max-knapsack problem and implement the approximation algorithm described in [8]. For more complex policies, we plan to adopt open-source solver, such as `lpsolve`.

We describe briefly how we map the problem into knapsack problem for a policy. Using policy I in Section 2.3 as an example, since the cost is a linear summation of computing, network and storage cost. We remove network and storage cost in the optimization process. Then, the problem fits into the min-knapsack problem. The approximation algorithm we implemented in `python` is a greedy-algorithm. The idea of the algorithm is to first sort n_{ij} in increasing order of $\frac{p_j}{\mu_j}$ then find the smallest list of n_{ij} that satisfy the capacity constraint.

Chapter 5

Evaluation

In the evaluation, we present experimental results on how well CloudGuide helps users choosing suitable cloud configuration. We conduct our experiments on two public clouds, Amazon EC2 and Hicloud.

5.1 Testbed application

We use RUBiS in our evaluation. RUBiS is a benchmark Internet service that implements core functionalities of an online auction site: selling, browsing and bidding [6]. RUBiS are implemented into three version: PHP, Java Servlets and Enterprise Java Bean (EJB). We choose RUBiS EJB version, which follows a three-tier model software architecture, containing a front-end web server, middle-tier business logic, and a back-end database. We host the application on three Dell R310 servers in local environment, with each tier deployed on a machine. The web server tier run on Apache 2.2, the business logic run on the JOnAS 5.3.0 application server, and the back-end database is MySQL 5.1.58. All tiers run on the Linux kernerl 2.6.18.

In all our experiments, we use the client workload generator that bundled with the application. Client workload generator allows us to decide the mix of operations and active sessions we use to exercise the web application. We use two typical workload in our experiments: 1.) Bidding-mix with 80% read requests and 20% write requests, 2.) Browse-only with 100% read requests. For simplicity we show results for experiments

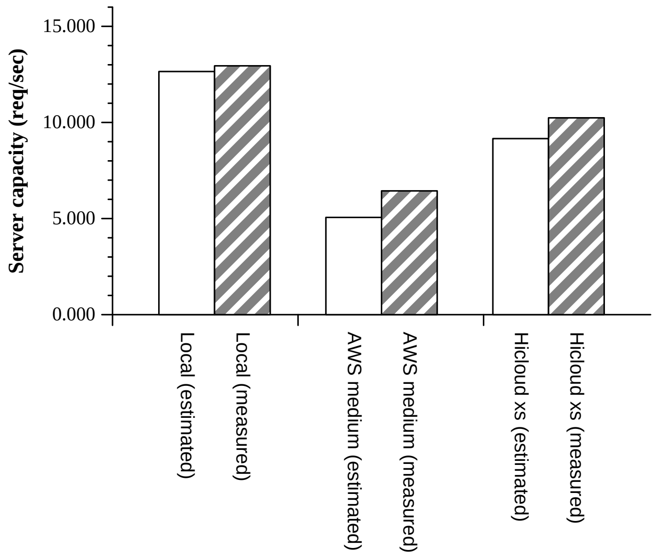


Figure 5.1: Estimated and measured server capacity for hosting RUBiS on local, Hicloud, and Amazon

conducted under bidding-mix workload by default. For simplicity we show results for experiments conducted under bidding-mix workload by default (CloudGuide performs better in browse-only mix due to simpler workload pattern).

5.2 Determining initial cloud configuration

We first evaluate the quality of the configurations suggested by CloudGuide for finding an initial cloud configuration when migrating legacy network application to the cloud. We show the predicted price and performance by CloudGuide before we migrate RUBiS from local environment to the cloud. The SLO of the average response time of the application is set to 2 seconds, peak request rate is set to 1000 active sessions, and the chosen policy is to minimize hosting cost. For cost constraints, we choose five types of instance from two cloud providers (identical to instances listed in figure 3.1) as candidates.

Figure 5.1 shows the estimated server capacity and cost of hosting RUBiS on local, Amazon and Hicloud by CloudGuide compared to the measured results. We show only the

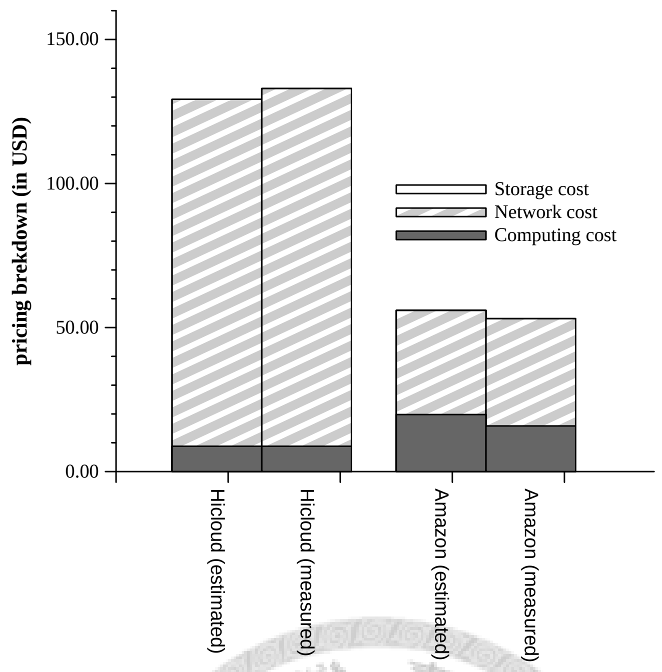


Figure 5.2: Estimated and measured cost for hosting RUBiS on Hicloud and Amazon

chosen instance type for each provider out of five total. For both providers, CloudGuide under-estimates the capacity by around 27% and 12% respectively. To satisfy performance requirements of peak request rate, CloudGuide generates a configuration of either three AWS medium instances or two Hicloud instances.

Figure 5.2 shows the accuracy of the estimated total computing cost for hosting RUBis on Hicloud and Amazon. We collected the application's resource usage on local cluster for one full day, calculated the estimated cost according to the cost calculation formula mentioned in Section 2.1. To compare the accuracy, we deployed the application on both Hicloud and Amazon EC2, and replayed workload mix similar to that observed on local cluster.

Here the results are in day granularity because day is the smallest granularity both provider display the cost before the final monthly bill. For both providers, storage cost is so small that it is hard to see in the figure. For both providers, the difference between

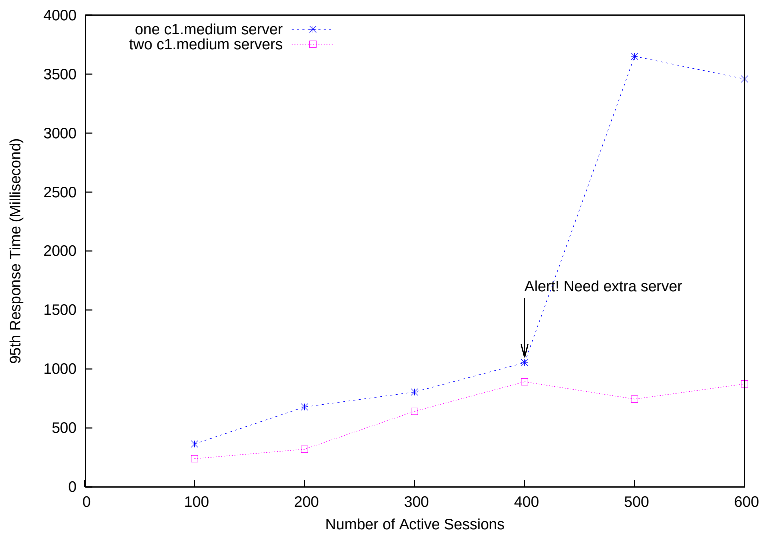


Figure 5.3: Invoke CloudGuide for dynamic provisioning when workload changes

estimated and measured cost is within 5%, which we think is accurate enough for users to make an informed decision. After considering both cost and performance requirements, CloudGuide suggests Amazon to best meet the policy. We observe during the experiment that the chosen configuration sustains the peak request rate requirement.

5.3 Dynamic provisioning when workload changes

Next, we study the case that leveraging CloudGuide to do capacity planning for dynamic provisioning. Users are able to leverage G/G/1 queuing model for scaling out (i.e., provide same type of new instance), and leverage G/G/m queuing modeling for scaling up (i.e., provide different type of new instance). Figure 5.3 depicts the result that we use CloudGuide to determine when and how many instances to *scale out* the application tier of RUBiS. As shown in the figure, as the active sessions increases over time, CloudGuide is triggered (assuming a monitoring module notices the resource shortage) at 400 active sessions. If the number of active sessions continue to raise and the tier is not scaled out as suggested by CloudGuide, SLO is violated.

Chapter 6

Related Work

Our work focuses on finding most suitable cloud provider meeting user-specified requirements to migrate legacy web applications with minimal effort from the user. Based on our best knowledge, we are the first to combine modeling techniques from queuing theory and cross-platform prediction on cloud provider selection problem with real deployment.

Previous work on calculating the cost of hosting an application in the cloud compared to locally provides partial solution to the same problem. Li et al. [19] outline the total-cost-of-ownership (TCO) and utilization cost to spend and build an in-house infrastructure compared to a cloud. They include the amortized cost of eight factors in their calculation and provide a web interface for user to query. Their system require users to input their estimated numbers for the data center size, which can be cumbersome. Our work differ from their work in that we also consider more fine-grained application resource need when determining how to migrate and where to migrate. Tak et al [27] apply Net Present Value (NPV) method to estimate and compare the cost of owning a local data center v.s. renting cloud resources. They take Amazon AWS as an example of cloud provider and include detailed cost calculation for deploying locally. Their work is complementary to ours as we compare among multiple cloud providers. Chen et al. determine which kind of company would benefit most from renting from cloud. Neither work addresses finding the most suitable cloud configuration that we address in this paper but their techniques are complementary to our work.

Work on benchmarking cloud performance also inspires our work. CloudCmp [18] developed benchmark tools to compare cost to performance ratio across four difference provider. Other works document their studies on migrating application with other characteristics [11, 30, 28]. All these work combined with ours can shed some light on helping users deciding whether and how to migrate their applications to the cloud.

Haijjat et al. [12] examine the problem of migrating client applications which are composed of several components into the cloud. They formulated their problem into an optimization problem by maximizing migration benefit while satisfying client-imposed policy constraints.

Conductor [31] considers the computation model based on MapReduce [9] and address resource management issues, such as dynamic pricing and resource diversity. Our work takes application resource requirement as an black-box and finds the best cloud provider based on the policy and requirements from users.

KingFisher [22] proposes cost-aware provisioning in the cloud; both formulate their problem as optimization problem. We use similar techniques but address the problem of provisioning for legacy web applications and include I/O cost.

Finally, Teregowda et al. [28] closely documented their migration process for a mature large-scale CiteSeer into the cloud and provided many insights on how application can be migrate in a hybrid cloud mode. Their work takes application-level knowledge into consideration for migration decision and we believe they can benefit from CloudGuide to explore what-if scenarios.

Chapter 7

Conclusion

We have taken the first step in estimating cost when hosting a web application in different cloud providers. Our work include: First, an initial survey of cloud providers and their respective offerings and pricing schemes. This survey helps us derive the suitable cost calculation function for user applications. Second, CloudGuide provides a systematic methodology to help user estimate cost of hosting their application in the cloud given different application QoS requirement. Specifically, we apply queueing model to help user predict suitable computing instance size that can sustain different capacity. Finally, CloudGuide helps users to provision dynamically when the workload shifts. We conducted experiments to show that even though fundamental unpredictability exist in cloud environment, applying queueing model is suitable to provision dynamic workload to accurately predict performance and estimate cost.

Bibliography

[1] Amazon web service. `http://aws.amazon.com`.

[2] Hicloud. `http://hicloud.hinet.net`.

[3] Ntu cloud. `https://www.cloud.ntu.edu.tw/`.

[4] Rackspace. `http://www.rackspace.com/`.

[5] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. Above the clouds: A berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[6] Cecchet, E., Chanda, A., Elnikety, S., Marguertie, J., and Zwaenepoel, W. Performance comparison of middleware architectures for generating dynamie web content. In *Proceedings of the 4th ACM/IFIP/USENIX International Middleware Confernce* (Rio de Janeiro, Brazil, June 2003).

[7] Cloudharmony. `http://www.cloudharmony.com/benchmarks`.

[8] Csirik, J., Frenk, J. B. G., Labbé, M., and Zhang, S. Heuristics for the 0-1 min-knapsack problem. *Acta Cybernetica 10*, 1--2 (September 1991), 15--20.

[9] Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation* (San Francisco, CA, December 2004).

[10] Doyle, R. P., Chase, J. S., Asad, O. M., Jin, W., and Vahdat, A. M. Model-based resource provisioning in a web service utility. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4* (Berkeley, CA, USA, 2003), USITS'03, USENIX Association, pp. 5--5.

[11] Garfinkel, S. L. An evaluation of amazon's grid computing services: Ec2, s3, and sqs. Tech. rep., Harvard University, 2007.

[12] Hajjat, M., Sun, X., and)David Maltz, Y.-W. E. S., Rao, S., Sripanidkulchai, K., and TawarmalaFni, M. Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud. In *Proceedings of SIGCOMM 2010* (New Helhi, INDIA, 2010).

[13] Hoste†, K., Phansalkar, A., Eeckhout, L., Georges†, A., John, L. K., and Bosschere, K. D. Performance prediction based on inherent program similarity. In *Proceedings of PACT'06* (Seattle, Washington, USA, September 2006).

[14] Kleinrock, L. *Queueing Systems Volume II: Computer Applications*. Wiley-Interscience, 1976, pp. 29--50.

[15] Kossmann, D., Kraska, T., and Loesing, S. An evaluation of alternative architectures for transaction processing in the cloud. In *The Proceedings of SIGMOD '10* (Indianapolis, Indiana, USA, June 2010), pp. 579 -- 590.

[16] Kundu, S., Rangaswami, R., Gulati, A., Zhao, M., and Dutta, K. Modeling virtualized applications using machine learning techniques. In *Proceedings of Eighth Annual International Conference on Virtual Execution Environments (VEE 2012)* (London, UK, March 2012), pp. 3--14.

[17] Li, A., Yang, X., Sandula, S., and Zhang, M. CloudCmp: Comparing Public Cloud Providers. In *Internet Measurement Confernce* (November 2010).

[18] Li, A., Zong, X., Zhang, M., Kandula, S., and Yang, X. Cloudprophet: Predicting web application performance in the cloud. Tech. rep., Duke University, 2011.

[19] Li, X., Li, Y., Liu, T., Qiu, J., and Wang, F. The method and tool of cost analysis for cloud computing. In *Proceedings of 2009 IEEE International Conference on Cloud Computing* (Bangalore, India, September 2009).

[20] Oprofile. `http://oprofile.sourceforge.net`.

[21] Schad, J., Dittrich, J., and Quiané-Ruiz, J.-A. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment 3*, 1 (2010).

[22] Sharma, U., Shenoy, P., Sahu, S., and Shaikh, A. A cost-aware elasticity provisioning system for the cloud. In *The Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS)* (June 2011).

[23] Shimizu, S., Rangaswami, R., Duran-Limon, H. A., and Corona-Perez, M. Platform-independent modeling and prediction of application resource usage characteristics. *J. Syst. Softw. 82*, 12 (Dec. 2009), 2117--2127.

[24] Stewart, C., Kelly, T., Zhang, A., and Shen, K. A dollar from 15 cents: Cross-platform management for internet services. In *In USENIX* (2008).

[25] Stewart, C., and Shen, K. Performance modeling and system management for multi-component online services. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI '05)* (Boston, MA, May 2005).

[26] sysstat. `http://sebastien.godard.pagesperso-orange.fr/t`.

[27] Tak, B. C., Urgaonkar, B., and Sivasubramaniam, A. To move or not to move: The economics of cloud computing. In *Proceedings of 3rd USENIX Workshop on Hot Topics in Cloud Computing* (Portland, OR, June 2011).

[28] Tergowda, P. B., Urgaonkar, B., and Giles, C. L. Citeseerx: A cloud perspective. In *Proceedings of HotCloud 2010* (Boston, MA, June 2010).

[29] Urgaonkar, B., Shenoy, P., Chandra, A., and Goyal, P. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1 (March 2008).

[30] Walker, E. Benchmarking amazon ec2 for high-performance scientific computing. *USENIX Login* (2008).

[31] Wieder, A., Bhatotia, P., Post, A., and Rodrigues, R. Conductor: Orchestrating the clouds. In *Proceedings of the 4th Workshop on Large Scale Distributed Systems and Middleware (LADIS)* (Zürich, Switzerland, July 2010).