$(EC)^2$ in EC2

Thomas A. Henzinger Anmol V. Singh Vasu Singh Thomas Wies
Damien Zufferey

IST Austria

Abstract

Cloud computing aims to give users virtually unlimited pay-per-use computing resources without the burden of managing the underlying infrastructure. We claim that, in order to realize the full potential of cloud computing, the user must be presented with a pricing model that offers flexibility at the requirements level, such as a choice between different degrees of execution speed, and the cloud provider must be presented with a programming model that offers flexibility at the execution level, such as a choice between different scheduling policies. In this paper, we present a framework called FlexPRICE (Flexible Provisioning of Resources in a Cloud Environment) where for each job, the user purchases a virtual computer with the desired speed and cost characteristics, and the cloud provider optimizes the utilization of resources across a stream of jobs from different users.

1 Introduction

Computing services that are provided by datacenters over the internet are now commonly referred to as *cloud computing.* Cloud computing promises virtually unlimited computational resources to its users, while letting them pay only for the resources they actually use at any given time. Cloud computing is generally a broad term that covers different services: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). For example, IaaS is provided by Amazon Elastic Compute Cloud (EC2) [1], PaaS is provided by Google App Engine [2], and SaaS is provided by Force.com [3].

In our work, we focus on IaaS. We believe that IaaS gives the cloud users maximal flexibility in terms of infrastructure to reserve and programs to execute. However, IaaS requires the user to bear responsibility for launching and terminating individual virtual machines as and when required. For instance, Amazon EC2 exposes a low-level interface to its datacenters where the user needs to decide which and how many virtual machines she should rent to execute a given job. This does not only pose a high burden on the user, but also leads to non-optimal utilization of the cloud: once a user rents a virtual machine, the cloud cannot run other computation on that machine. Similarly, the existing pricing models are too rigid to foster good utilization. For instance, both Amazon EC2 [1] and Microsoft Windows Azure [4] charge fixed prices for compute usage, storage, and data transfer.

A common analogy to utility computing is the modern day electricity distribution. A century ago, people relied on household generators to meet their individual electricity demands. Today, people meet their electricity demands from the common electricity grid. A user may plug in as many appliances as she wants, and does not care about how the electricity company meets the demand. The electricity company optimizes the distribution of electricity and charges the user according to the consumption. In this respect, management of computing resources in utility computing is in a nascent stage. While some cloud providers (like Google App Engine [2] and Microsoft Azure [4]) restrict the kind of programs that can be executed on their data centers, others (like Amazon EC2 [1]) require the user to optimize the price of computation by renting the best possible combination of machines. This optimization is not trivial for large programs, and a poor choice by the user may lead to two undesirable outcomes: the user pays a lot more than the optimal amount, and the cloud resources are poorly utilized. Consider a simple example where a user wishes to use Amazon EC2 to apply a sequence of image transformations to all images in her image database residing in Amazon S3 services in the geographical location G. If the user has a budget or a deadline, she is faced with multiple questions:

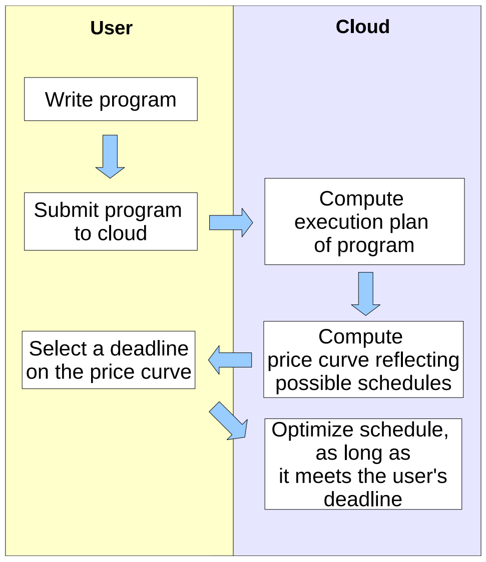
- *What size of the instances should she rent?* The user may rent multiple small instances or a single high-cpu instance.
- *In which geographical region should the user rent the instances?* The price of an EC2 instance depends on the geographical location. Moreover, transferring images from one geographical location to another can be expensive. Should the user rent instances in location G itself, or in a different location?
- *How should she handle failures?* Should she checkpoint the intermediate images obtained or should she replicate the image transformations on multiple EC2 instances?

Our goal is to build the next generation of resource management in cloud computing. We propose "Flexible Provisioning of Resources In a Cloud Environment" (FlexPRICE) where the cloud (provider) and the users build a symbiotic relationship. We believe that such a symbiotic relationship would allow to exploit the massive computing services offered by a cloud more efficiently. Instead of renting a set of specific resources, the user simply presents the job to be executed to the cloud. The cloud has an associated pricing model to quote prices of the user jobs executed. In FlexPRICE we assume that each computation node has a computation price and possibly an initial setup price. Additionally, each link may have an associated data transfer price. The pricing models in FlexPRICE also allow to discount delayed execution of jobs. The cloud works out multiple possibilities to execute the job, then presents to the user a *price curve* which is a relation between time and price. For the user, the price curve represents the trade-off between the delay of job execution and the price for the execution. A fast computation, which can be due to high end processors or highly parallelized computation, may price more than slow or delayed computation. For the cloud, the price curve represents the optimizations the cloud can apply to the job while executing it. For example, longer execution times provide more possibilities for scheduling and using intermediate idle periods of the cloud. On the other hand, shorter execution times are more rigid in terms of execution requirements. The user observes the price curve and chooses a point on the curve according to her requirements on the latest completion time of the job (deadline) and the maximum price she is willing to pay (budget). After the user expresses her requirement, the cloud is bound to schedule the job such that the users' requirements are satisfied.

```
image.job:
input = get_bucket b1
output = map (t) (input) (8)
reduce (put_bucket b2) (output)

task t:
duration 10
memory 4
convert -paint 10 $1 $2
```

(a) User program


User
Cloud
Write program
Submit program to cloud
Compute execution plan of program
Compute price curve reflecting possible schedules
Select a deadline on the price curve
Optimize schedule, as long as it meets the user's deadline

(b) Workflow of FlexPRICE

Figure 1: System description

2 Design Principles

The design of FlexPRICE is motivated by the following principles:

- *A simpler view of the cloud to the user.* We advocate a method where a user submits a user program, called a job, to the cloud for execution. A job corresponds to *what* has to be done, and a schedule, which is computed by the cloud, corresponds to *how* the job is done. The cloud generates multiple schedules, where each schedule has a corresponding finish time, a price, and a probability of failure. In other words, letting the cloud optimize the computing resources allows the user to transparently view an abstraction of the cloud.

- *Optimization of resource allocation by the cloud.* Many jobs of different users are simultaneously executed in a cloud. A cloud is in a position to optimize the allocation of computing resources depending upon the current utilization. A cloud can choose from a range of different pricing models and scheduling policies as required at a particular time. This enables the cloud to adapt itself to the incoming stream of jobs from all users. For example, in peak hours, the cloud can postpone the execution of a job to later periods as long as it satisfies the requirements of the user. It is unrealistic to expect that the individual user makes an optimal choice when allocating her resources. The information that is necessary to make such an optimal choice is simply not available to the user.

3 Implementation

We have developed a prototype of FlexPRICE . We used Eucalyptus to create and manage virtual machines on our campus at IST Austria. A user job is an archive consisting of the user program and any input data that the user wants to send with the program. Figure 1(a) shows an example of a map/reduce job. The first line of the job description specifies an inherent task `get_bucket` which fetches all files in a bucket of the associated database. With Amazon EC2, the corresponding database service is Amazon S3. With Eucalyptus, such a database is provided by the Walrus service. The second line of the job description defines a mapping task that runs 8 instances of task `t`. Task `t` describes an image conversion operation. The task description consists of resource requirements (here given as functions in terms of the size of the input image), and a command that invokes the binary executable `convert` associated with the task. The last line of the job description defines a reduce operation which puts back the converted images into the database using the inherent task `put_bucket`.

We used Python to create a web interface for users to submit jobs to our cloud setup. When a user submits a job, the web interface interacts with the scheduler to produce different schedules for the job according to the current state of the cloud. In our example the scheduler may produce, among others, a schedule that runs all 8 instances of task `t` in parallel on 8 fast machines, resulting in an early finish time but a high price, and an alternative schedule that runs all 8 tasks on a single slow machine, resulting in a late finish time but a lower price. Once a set of schedules has been computed, they are used to produce a price curve which is provided to the user. Both the scheduler and the price curve generator are implemented in OCaml. When the user is presented with the price curve, she is free to choose any point on the curve.

Once a price submitted by a user has been confirmed by the cloud, the cloud manages the execution of the job on the virtual machines. This requires transfer of the tasks descriptions and the initial data to the virtual machines that are determined by the chosen schedule. The actual task execution is taken care of by a job daemon residing on each virtual machine. The job daemon is written in C++. It manages all tasks that have been scheduled on the particular virtual machine. This subsumes the transfer of input data for each task from other virtual machines and the execution of the task at its scheduled start time. The job daemon also monitors the execution and ensures that the task does not exceed its resource bounds. The workflow of FlexPRICE is shown in Figure 1(b).

4 Conclusion

We have motivated a vital requirement of a suitable abstraction between the users of the cloud and the cloud provider. This transparency establishes a symbiosis between the cloud and its users. We introduced a novel framework FlexPRICE which achieves this transparency and described a prototypical system that implements this framework.

References

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

[2] Google App Engine. `http://code.google.com/appengine/`.

[3] Enterprise cloud computing. `http://www.salesforce.com/platform`.

[4] Windows Azure Platform. `http://www.microsoft.com/windowsazure`.