FlexPRICE: Flexible Provisioning of Resources in a Cloud Environment

Thomas A. Henzinger Anmol V. Singh Vasu Singh Thomas Wies Damien Zufferey
IST Austria
A-3400 Klosterneuburg, Austria
{*tah,anmol.tomar,vasu.singh,thomas.wies,damien.zufferey*}*@ist.ac.at*

***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 such a flexible framework, with each job, the user purchases a virtual computer with the desired speed and cost characteristics, and the cloud provider can optimize the utilization of resources across a stream of jobs from different users.**

We designed a flexible framework to test our hypothesis, which is called FlexPRICE (Flexible Provisioning of Resources in a Cloud Environment) and works as follows. A user presents a job to the cloud. The cloud finds different schedules to execute the job and presents a set of quotes to the user in terms of price and duration for the execution. The user then chooses a particular quote and the cloud is obliged to execute the job according to the chosen quote. FlexPRICE thus hides the complexity of the actual scheduling decisions from the user, but still provides enough flexibility to meet the users actual demands. We implemented FlexPRICE in a simulator called PRICES that allows us to experiment with our framework. We observe that FlexPRICE provides a wide range of execution options —from fastand expensive to slow and cheap— for the whole spectrum of data-intensive and computation-intensive jobs. We also observe that the set of quotes computed by FlexPRICE do not vary as the number of simultaneous jobs increases.

I. 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. We question that the existing cloud computing solutions can effectively deliver on this promise. Cloud computing services such as Amazon EC2 [1] and Google App Engine [2] are built to take advantage of the already existing infrastructure of their respective company. This development leads to non-optimal user interfaces and pricing models for the existing services. It either puts an unnecessary burden on the user or restricts the class of possible application. 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 [3] charge fixed prices for compute usage, storage, and data transfer. Recently Amazon added the possibility to bid for instances whose price depends on supply and demand [4]. Therefore, a flexible pricing model that, for example, discounts compute usage during non-peak hours seems adequate.

Motivation. 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. 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. A fast computation, which can be due to high end processors or highly parallelized computation, may price more than slow or delayed computation. 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.

The design of FlexPRICE is motivated by the following principles:

- *A simpler view of the cloud to the user.* Today's cloud

services vary in the abstraction presented to the user. On the one hand, services like Amazon EC2 provide the users with complete freedom to control and configure the entire software stack and thus do not limit the type of applications that can be hosted. On the other hand, Google App Engine, Force.com provide highly application-specific cloud services. Thus, the user is either left with a responsibility to optimize execution as in the first case, or is limited in the type of applications she can run on the cloud as in the second case.
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 and a price. 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. Even at the individual user level it is unrealistic to expect a user to make optimal choice in term of resources allocation. Since one of the selling point of cloud computing is hiding the inner complexity of a datacenter the information necessary to make optimal choices are not provided.

Simulation results. Based on these principles, we develop "Simulator for Provisioning resources in a Cloud Environment" (PRICES). We use PRICES to extensively study FlexPRICE. We study how a mix of different scheduling heuristics find the intuitive schedules for a range of data intensive and computation intensive jobs. We show that a cloud using FlexPRICE helps the users and the cloud. We conduct preliminary studies on the effect of pricing models of the resources on the economic advantage of a cloud. Based on our study, we draw the following conclusions.

- *FlexPRICE discovers a range of scheduling possibilities.* We consider common patterns of data intensive and computation intensive jobs. In a data intensive MapReduce job, the data is first processed independently at many locations (mappers), and later the results are merged together at one location (reducer). We study the set of quotes that a cloud using FlexPRICE gives to the user. We observe that while the mappers always execute in parallel at locations where the initial data sits, the location of the reducer is sensitive to the completion time that the user expects.
- *FlexPRICE creates robust price curves.* Given a job to be executed on a cloud, we say that the price curve is *robust* if it does not change if the cloud is running more jobs simultaneously. In our simulation experiments we model the choice of users as a normal distribution. We compare the flexible scheduling and static scheduling. The results for varied number of jobs showed that a flexible scheduling generated a more robust price curve.

II. The Formal Model

We give a formal description of our model. We start with a description of a cloud infrastructure. Then, we formalize user programs (jobs) and schedules of jobs on the cloud infrastructure.

Cloud.: A *cloud* is a term used broadly for the infrastructure of a datacenter – cpus, network, and other peripherals. In our model, we represent a *cloud* as a fully connected graph of networked computation nodes. We assume that there exists a communication link between each pair of nodes. We also assume that each link has an individual bandwidth and the data transfer on one link does not affect the other links. This assumption allows us to separate the orthogonal issue of distribution of total bandwidth across the links from our work.

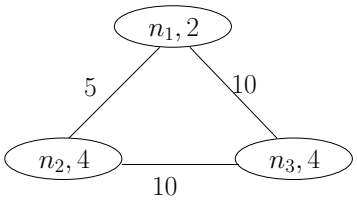
We model a datacenter infrastructure as a set of cpu nodes. A node n corresponds to a computing entity like a physical or a virtual machine. An edge e is a communication link between two nodes. Formally, we define a *cloud* as $C = \langle N, E, S, B, \pi \rangle$ where N is the set of cpu nodes, $E = N \times N$ is the set of communication links between the cpu nodes, the function $S : N \to \mathbb{N}$ represents the speed of the nodes in the cloud, the function $B : E \to \mathbb{N}$ represents the bandwidth of every edge in the cloud, and the pricing model π determines how users are charged for using the cloud. The pricing model if formally defined later.

Example. Figure 1 shows an example of a cloud. The cloud is depicted by the graph in the upper part of the figure. The numbers on the edges represent the bandwidth of communication links. The nodes contain an identifier and their respective speed.

Job.: Users submit jobs to be executed on the cloud. We describe a job as a graph consisting of independent tasks as nodes and data transfers between them as edges.

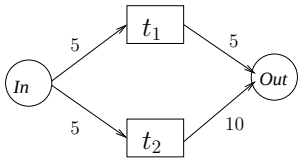
Formally, a *task* t corresponds to a piece of computation in a user program. An object o is a piece of data that is transfered from one task to another. A job is a directed acyclic graph (DAG) $J = \langle T, O, D, Z, \mathit{data} \rangle$ where

- $T = T_d \cup T_c$ is the set of tasks, where T_d is a set of data tasks T_c is a set of computation tasks,
- $O \subseteq T \times T$ is the set of objects between the tasks,



$$\begin{aligned}\pi: \quad & cc(t)(n) = D(t) \cdot S(n) \\ & tc(o)(e) = 0 \\ & sc(n) = 0 \\ & df(\sigma) = 1\end{aligned}$$

Figure 1. An example cloud C_e.



T	D	*data*(C)
In	-	n_1
t_1	12	-
t_2	24	-
Out	-	n_2

Figure 2. A job J_e. The data tasks are shown in circles and computation tasks in squares.

- $D : T_c \to \mathbb{N}$ gives the computation duration of a given task on a unit cpu in time units,
- $Z : O \to \mathbb{N}$ gives the size of a given object in terms of the unit data size.
- *data* $: C \to T_d \to N$ takes as input a cloud, and gives a mapping of data tasks to the nodes in the cloud.

Intuitively, the $data$ function captures the constraints of executing the persistent reads and writes of a job at specific nodes in the cloud.

Example. Figure 2 shows an example of a job J_e. The numbers on the edges represent the size of the objects. The adjacent table lists the duration of the computation tasks and the data constraints of the data tasks with respect to the cloud C_e shown in Figure 1.

Schedule: A schedule maps the set of tasks in a job to nodes and time intervals of execution on the nodes. Formally, given a job J and a cloud C, a *schedule* $\sigma : T \to N \times \mathbb{Q}^+$ assigns tasks of J to the nodes in C, at specific intervals.

In general, a job can be executed on a cloud with many different schedules. Let Σ be the set of all schedules of the job J on cloud C. We define a *scheduling algorithm* by the function $A : C \times J \to 2^{\Sigma}$. Intuitively, a scheduling algorithm takes a cloud and a job, and returns a set of possible schedules.

A *well formed schedule* is one in which the following hold: each task is scheduled exactly once, each task is scheduled for a duration not less than its computation duration on the given cpu, and all task dependencies (which may be a partial order) are respected.

The execution time of a schedule is given by the function *duration* $D : \Sigma \to \mathbb{Q}^+$. The duration of a given schedule can be derived from the duration of each task in the job, the size of each object, and the bandwidth of each link.

Pricing Model: A *pricing model* π of a cloud C determines how a user is charged for executing a schedule on the cloud. Formally, π is a tuple $\langle cc, tc, sc, td \rangle$ where

- the function $cc : T \to N \to \mathbb{Q}$ determines the *computation cost* for executing a given task on a given node,
- the function $tc : O \to E \to \mathbb{Q}$ computes the *data transfer cost* for transferring a given object over a given communication link.
- the function $sc : N \to \mathbb{Q}$ determines the *setup cost* for each node
- the function $td : \Sigma \to \mathbb{Q}$ denotes the *time discount factor* for a given schedule.

Example. The lower part of Figure 1 shows the pricing model of cloud C_e. The computation costs of a task on a node are linear in the duration of the task, scaled by a constant node-dependant factor. There are no transfer, no setup costs, and no time discounting.

Given a pricing model π, we define a function *price* $P_\pi : \Sigma \to \mathbb{Q}$ that gives the price incurred by the cloud for executing a schedule for a given job $J = \langle T, O, D, Z, \mathit{data} \rangle$. The price of a schedule is given by the sum of its total computation costs, total data transfer costs, and total setup costs, scaled by the time discount factor. Formally, the function P_π is defined as follows:

$$P_\pi(\sigma) = td(\sigma) \cdot P'_\pi(\sigma)$$

where

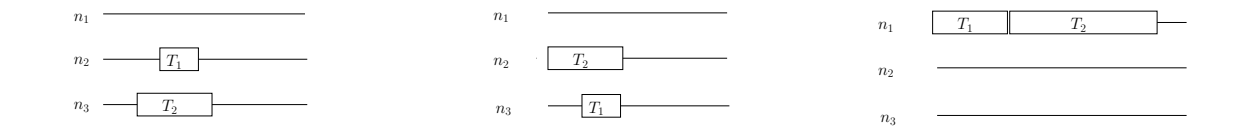
$$\begin{aligned}P'_\pi(\sigma) = & \sum_{t \in T} cc(t)(\sigma(t)_{\#1}) \\ & + \sum_{(t_1,t_2) \in O} tc((t_1,t_2))(\sigma(t_1)_{\#1}, \sigma(t_2)_{\#1}) \\ & + \sum_{n \in un(\sigma)} sc(n)\end{aligned}$$

Here $x_{\#i}$ denotes projection of a tuple x to its i-th component. The set of *used nodes* $un(\sigma)$ of a schedule σ is defined by $un(\sigma) = \{\sigma(t)_{\#1} \mid t \in T\}$.

Example. Figure 3 shows some sample schedules of the job J_e on cloud C_e with their associated durations and prices.

Price Curve: Different schedules have different completion times and different prices. We say that two schedules σ_1 and σ_2 are equivalent if $P_\pi(\sigma_1) = P_\pi(\sigma_2)$ and $D(\sigma_1) = D(\sigma_2)$. Given two schedules σ_1 and σ_2, we say that σ_2 is *worse than* σ_1 if σ_1 is not equivalent to σ_2, $P_\pi(\sigma_1) \leq P_\pi(\sigma_2)$, and $D(\sigma_1) \leq D(\sigma_2)$. Let Σ be a set of schedules of a given job on a given cloud. We define a function *mono* such that given a set Σ of schedules, the function $\Sigma' = \mathit{mono}(\Sigma)$ gives the largest subset of schedules in Σ such that for every two schedules σ_1 and σ_2 in Σ', σ_1 is not worse than σ_2.

We now make formal our notion of a price curve. The main purpose of the price curve is to hide the details about



$$D(\sigma_1) = \max(\tfrac{5}{5} + \tfrac{12}{4}, \tfrac{5}{10} + \tfrac{24}{4} + \tfrac{10}{10}) = \max(4, 7.5) = 7.5$$
$$P_\pi(\sigma_1) = (12 \times 4) + (24 \times 4) = 48 + 96 = 144$$

(a) Schedule σ_1

$$D(\sigma_2) = \max(\tfrac{5}{10} + \tfrac{12}{4} + \tfrac{5}{10}, \tfrac{5}{5} + \tfrac{24}{4}) = \max(4, 7) = 7$$
$$P_\pi(\sigma_2) = (12 \times 4) + (24 \times 4) = 48 + 96 = 144$$

(b) Schedule σ_2

$$D(\sigma_3) = (\tfrac{12}{2} + \tfrac{24}{2} + \tfrac{5}{10}) = 18.5$$
$$P_\pi(\sigma_3) = (12 \times 2) + (24 \times 2) = 24 + 48 = 72$$

(c) Schedule σ_3

Figure 3. Some possible schedules for the job J_e on the cloud C_e. The calculation of the price and the duration of the schedules is given.

the scheduling process and the internal working of the cloud from the user, but still offer the user some control over the execution of her job.

Let C be a cloud and J be a job. Let further Σ be a set of (monotonic) schedules for J on C and let be t_{min} the duration of the fastest schedule in Σ. A *price curve* for the set Σ is a function $f : (t_{min}; \infty) \to \mathbb{Q}^+$. A price curve has the property that for any price p the user is willing to pay, there is a schedule in Σ that can meet the corresponding time $f^{-1}(p)$. For pragmatic reasons the curve should be monotonically decreasing, because the user expects to pay more only if the quality of service increases. The concrete choice of the shape of a price curve is more of an economic decision than an engineering one. We therefore do not further constrain the price curve in our formal model. In section III we discuss how we obtain a price curve from a given set of monotonic schedules.

III. Simulation Framework

We develop a simulator PRICES to study our model. PRICES provides a simple syntax to create clouds and jobs. PRICES also allows to randomly generate clouds and jobs, to schedule jobs and sequences of jobs on clouds according to different user distributions, and to compare different pricing models.

Workflow of the Simulator.: A typical work flow for using our simulator is as follows. One first randomly generates a cloud of specified size and heterogeneity. Then one randomly generates a sequence of jobs that are released according to a Poisson process of a given rate. The size and density of the jobs' task graphs follow a Pareto distribution. The simulator then simulates the execution of the generated job sequence on the cloud according to a chosen pricing model and user distribution.

The execution of a job on the cloud is simulated as follows. The simulator first computes a set of possible schedules with respect to the current state of the cloud, i.e., taking into account all jobs that have already been scheduled. The candidate schedules are computed by sampling schedule algorithms that produce different schedules depending on a free parameter such as a user-specified deadline or maximal cost. Each computed schedule σ induces a point in the time/price plane that is given by the duration of the schedule and its price according to the chosen pricing model. The simulator then constructs a price curve that fits the computed set of points. The simulator then chooses a point on the constructed price curve corresponding to the chosen user distribution. Then the schedule from the sampled set of schedules that is closest to the chosen point on the price curve is selected and executed on the cloud.

Scheduling Heuristics.: Most of the optimization problems in the domain of scheduling are NP-hard. For example, finding time-optimal, respectively, cost-optimal schedules, and finding the cheapest schedule for a given deadline, respectively, the fastest schedule for a given budget are all shown to be NP-hard problems. Instead of computing optimal schedules we therefore employ scheduling heuristics that produce good approximations of the optimal schedules. Based on the existing literature on scheduling heuristics, we developed the following schedulers: (i) a *greedy scheduler* which takes two tuning parameters α and β and at every step chooses the node which minimizes the sum of α times the price of the assignment and β times the duration of the assigned task. Tuning the parameters α and β gives a whole spectrum of greedy schedules from fast expensive schedules to slow cheap schedules (ii) a *clustering scheduler* based on the Dominant Sequence Clustering (DSC) algorithm [5] for scheduling task DAGs on multiprocessors, (iii) a *deadline division scheduler* [6] which takes a deadline as a parameter and applies a distribution of the deadline over task partitions, where a partition is a set of connected tasks in the job, (iv) a *genetic scheduler* which takes as input a budget and finds schedules with price less than the budget.

Constructing a Price Curve.: In order to construct the price curve from the computed set of sample schedules, we first remove all schedules from the set that violate the monotonicity property. We then fit a price curve of a predefined shape to the points that are determined by the remaining set of schedules Σ. The price curves we use in our simulation framework have the following shape:

$$f(t) = a(t - t_{min})^{-b} \quad \text{where } a, b > 0$$

Here t_{min} is the minimal duration of all schedules in Σ. The curves of this shape have two important properties: (1) they converge to ∞ for $t \to t_{min}$, (2) they converge to 0 for $t \to \infty$. Property (1) enforces that the user cannot choose a deadline that is not met by the fastest schedule. Property (2) comes from the observation that the computing power has been steadily increasing over the years. Thus, in the limit, computation is free. The parameters a, b are chosen such that the distance between the resulting curve f and the given sample points is minimized subject to the condition that for all schedules $\sigma \in \Sigma$, $f(D(\sigma)) \geq P_\pi(\sigma)$. The price curve that our simulator constructs is thus a conservative approximation of the actual prices that can be guaranteed for the chosen pricing model.

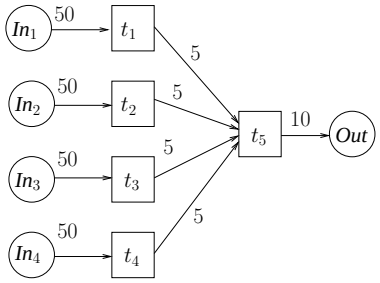
User Distributions.: A user in our simulator is a random variable in the $[0; 1]$ interval. The random variable determines a point on the line connecting the points corresponding to the fastest and the cheapest schedules in the sample set. This point is projected on the constructed price curve to determine the actual choice of the user. A user value close to 1 will choose a high-priced fast schedule, while a user value close to 0 will choose a low-priced slow one. Our simulation of the user behavior ensures that the user distribution is independent of the shape of the price curve. The following user distributions are implemented in our simulator: (1) *Normal distribution* where most users will pick points close to the middle of the price curve. (2) *'Inverse' normal distribution* which is the inverse of the normal distribution, i.e., most users will pick either very cheap and slow or very expensive and fast schedules. It is suitable for scenarios where most jobs can either be planned far in advance or are unexpected jobs that need to be executed quickly at a potentially much higher price. (3) *Uniform distribution* over $[0; 1]$. (4) *Hasty distribution* which model users who always return 1, i.e., the distribution is a Dirac delta at 1. The purpose of the hasty user distribution is to simulate a scenario with the extreme behavior where all users want a fast execution of their jobs. (5) *Greedy distribution* which is similar to the hasty distribution, but returns always 0. This distribution models the extreme case where all users prefer low-priced slow schedules.

IV. Experiments

We now describe our experiments using PRICES.

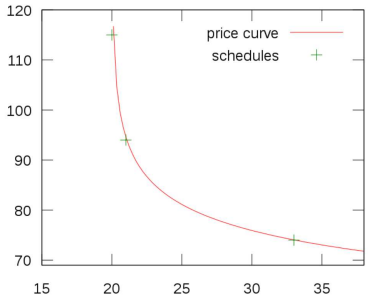
Analysis of Schedules.: We construct jobs which model different scenarios that exist in practice. We want to show that our scheduling heuristics give reasonable and intuitive schedules. We model scenarios which are data or computation intensive. We describe our observations with the data intensive jobs. We consider a cloud with nine nodes $n_1 \ldots n_9$. The bandwidth of all edges is 5. The nodes n_1 to n_4 have a speed of 1, nodes n_2 to n_8 have a speed of 2, and node n_9 has a speed of 3. The pricing model consists of a computation price per unit time equal to the speed on the nodes, and a transfer price on the edges per unit time equal to the bandwidth of the edges. We consider a MapReduce job as shown in Figure 4(a). The tasks $t_1 \ldots t_4$ are the mappers, and the task t_5 is the reducer. While the original data resides on different nodes, the final result has to be put at a specific node in the cloud. Note that the objects read by the mappers are of large size compared to the objects produced by the mappers. We observe that in all schedules, every mapper task is executed at the place where the initial data resides. This is because the mappers read large objects, and hence moving the data is both time-consuming and expensive. Note that if the pricing model does not include transfer cost, then the cheapest schedule would execute all mappers sequentially on the node with the lowest price. The fastest schedule executes the reducer on the fastest node (n_9). The cheapest schedule schedules the reducer on one of the cheapest nodes (n_1). An intermediate schedule, which is nearly as fast as the fastest schedule but much less expensive, schedules the reducer on node n_5 where the output is finally expected. This saves both transfer time and cost. For data intensive scenarios, we obtain schedules which minimize the transfer of data from one node to another, as long as the data transfer helps to parallelize computation and the user is willing to pay the price for the fast computation.

Benefit of FlexPRICE *to Users and the Cloud.:* Conventional resource allocation schemes do not provide a choice on the price curve to the user. For example, a fastest execution scheme would minimize the finish time of a job, and a cheapest execution scheme would minimize the price of a job. Given a job to be executed on a cloud, our framework returns a range of schedules as described by the price curve. We model the specific choice of a user by different distributions on the range of prices in the price curve. The cloud available for the next job depends on the choice of the first user. Intuitively, as the cloud becomes more constrained, the number of possible schedules for a given job reduces. This, in turn, leads to a shift in the price curve. The shift in the price curve can be thought of as interference to the price a user has to pay for a job in the presence of other jobs. In other words, a cloud offers robust price curves if the shift in the price curve with respect to the price curve of the empty cloud is small. We show that FlexPRICE gives robust price curves. Note that robust price curves imply that a cloud can meet stringent deadlines

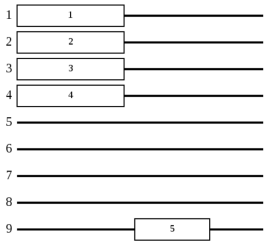


T	D	$data(C)$
In_1	-	n_1
In_2	-	n_2
In_3	-	n_3
In_4	-	n_4
Out	-	n_5
t_1	10	-
t_2	10	-
t_3	10	-
t_4	10	-
t_5	20	-

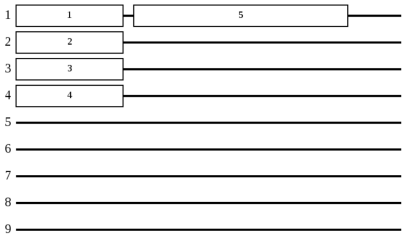
(a) Map reduce job



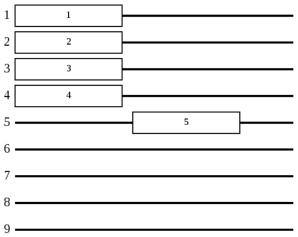
(b) Price curve for data intensive MapReduce job



(c) The fastest schedule

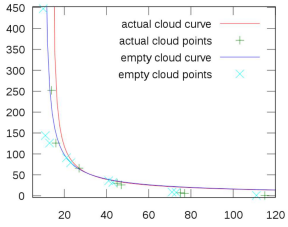


(d) The cheapest schedule

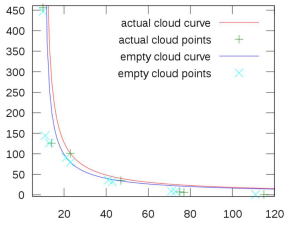


(e) An intermediate schedule

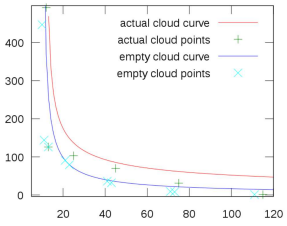
Figure 4. Analysis of a MapReduce job



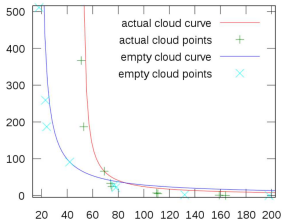
(a) Hasty: 20 jobs



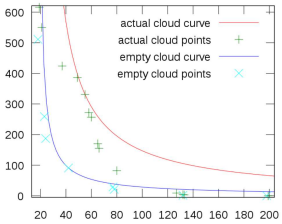
(b) Normal: 20 jobs



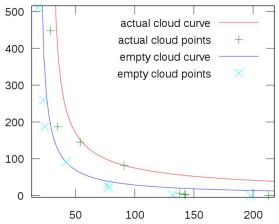
(c) Inverse normal: 20 jobs



(d) Hasty: 80 jobs



(e) Normal: 80 jobs



(f) Inverse normal: 80 jobs

Figure 5. Price curve shifts

and thus satisfy more users. We create a random cloud using PRICES. We execute a sequence of 100 randomly generated jobs. For each job, we find the price curve if the job is executed on the empty cloud. We compare this price curve with the price curve obtained if all previous jobs are scheduled according to a fixed policy which always chooses the fastest schedule. We then compare the empty price curve where the user chooses a point on the price curve using two different distributions: normal and inverse normal. Figure 5 shows the shift in the price curve with respect to the price curve on the empty cloud after 20 and 80 jobs. We observe that under the hasty execution scheme, the price curve drastically shifts to the right as the number of jobs increases. On the other hand, the price curves under normal and inverse normal distributions gradually shift both in price and in time. The flexibility offered by these user distributions regulates the shift of the price curve.

V. Concluding Remarks

Scheduling is fundamental to the achievement of high performance in parallel and distributed systems. The classic work on multiprocessor scheduling dates back to 1977 [7], where the problem of scheduling a directed acyclic graph of tasks to two processors is solved using network flow algorithms. As multiprocessor scheduling of directed task graphs to find an optimal schedule is an NP-complete problem, heuristics in scheduling have been employed. In utility computing and grid computing, scheduling optimization problems with given user requirements of deadline and budget have been studied [6], [8].

Various programming models for data oriented and computation oriented programs have also been developed. Data oriented programming models include the MapReduce framework [9], [10]. These models minimize the transfer of data, and maximize the parallel execution of independent tasks. The MapReduce model has been widely studied. Similar models which extend or present alternatives to MapReduce include DryadLINQ [11], Pig Latin [12], Sawzall [13]. Systems that improve the performance of the original MapReduce implementation have been developed [14]. While MapReduce and its derivatives are widely used in Internet-scale tasks, these models are not applicable to scientific tasks, which are inherently computation intensive. A plethora of programming languages, including Chapel [15], Fortress [16], and X10 [17] have been built for scientific computing. These languages have built-in constructs for parallelizing loops, asynchronous calls and parallel data structures. These languages are together called the High Performance Computing (HPC) languages [18]. In an effort to unify these programming models, our work is partly inspired by these models. However, our work focuses on allocation of resources across tasks using scheduling heuristics. The question of creating a job from a given user program remains to be answered.

Although in its inception, the cloud computing research has gained momentum recently. There are various platforms and technologies available which differ in the services made available to the users. Amazon EC2 [1], Google App Engine [2], Microsoft Azure [3], and Force.com [19] are few popular services. Cloudera [20] offers commercial support to Hadoop enterprise-level users. The various research perspectives in this direction can be observed in Microsoft's workshop on declarative datacenter [21]. A perspective of the current trends in cloud computing was also studied by Armburst et al. [22]. All these efforts study datacenter management as a programming problem and related issues. This paper presents a flexible and transparent pricing model such that the burden of price optimization and scheduling is taken off the user, and the cloud resources are efficiently utilized.

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. We also designed and implemented a simulation framework PRICES in which we model clouds, the pricing models and the user jobs. We validate the usefulness of our proposal across various experiments representing realistic and plausible scenarios. In particular, we used our simulation framework to study the robustness of price curves in a scenario where a cloud executes many simultaneous jobs. Our simulation results have encouraged us to start implementing FlexPRICE in an actual system.

References

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

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

[3] "Windows Azure Platform," http://www.microsoft.com/windowsazure.

[4] "Amazon EC2 Spot Instances," http://aws.amazon.com/ec2/spot-instances.

[5] T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," *IEEE Transactions on Parallel and Distributed Systems*, pp. 951–967, 1994.

[6] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow application on utility grids," in *International Conference on e-Science and Grid Computing*. IEEE Computer Society, 2005, pp. 140–147.

[7] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Transactions on Software Engineering*, pp. 85–93, 1977.

[8] J. Yu and R. Buyya, "A budget constraint scheduling of workflow applications on utility grids using genetic algorithms," in *Workshop on Workflows in Support of Large-Scale Science*, 2006.

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, pp. 107–113, 2008.

[10] "Apache Hadoop," http://wiki.apache.org/hadoop.

[11] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, and J. Currey, "DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language," in *USENIX Symposium on Operating Systems Design and Implementation*, 2008, pp. 1–14.

[12] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A not-so-foreign language for data processing," in *ACM SIGMOD Conference*, 2008, pp. 1099–1110.

[13] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the data: Parallel analysis with Sawzall," *Scientific Programming*, pp. 277–298, 2005.

[14] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *USENIX Symposium on Operating Systems Design and Implementation*, 2008, pp. 29–42.

[15] B. L. Chamberlain, D. Callahan, and H. P. Zima, "Parallel programmability and the Chapel language," *International Journal of High Performance Computing Applications*, pp. 291–312, 2007.

[16] G. L. Steele, "Parallel programming and parallel abstractions in Fortress," in *IEEE PACT*, 2005, p. 157.

[17] P. Charles, C. Grothoff, V. A. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar, "X10: An object-oriented approach to non-uniform cluster computing," in *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2005, pp. 519–538.

[18] M. Weiland, "Chapel, Fortress and X10: Novel languages for HPC," http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0706.pdf, 2007.

[19] "Enterprise cloud computing," http://www.salesforce.com/platform.

[20] "Cloudera," http://www.cloudera.com/.

[21] K. Bhargavan, A. Gordon, T. Harris, and P. Toft, "The Rise and Rise of the Declarative Datacentre," Microsoft Research, Tech. Rep. MSR-TR-2008-61, 2008, available at: http://research.microsoft.com/pubs/70575/tr-2008-61.pdf.

[22] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," University of California at Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009, available at: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.