ISSN:2231-0711
www.ijcset.net

Flexible Provisioning and Integrated Load Balancing of Resources in the Cloud

Pranav Kurbet, Poornima A.B

Dayananda Sagar College of Engineering, Bangalore

pranavkurbet@yahoo.com.

poornima.dsce@gmail.com.

Abstract— **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.**

Our goal is to build the next generation of resource management in cloud computing. We propose "Flexible Provisioning and Integrated Load Balancing of Resources in Cloud" 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.

Keywords— **Greedy Scheduler, Deadline Division Scheduler, Central Monitor, Resource Allocator, Cloud Infrastructure.**

I. Introduction

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

We question that the existing cloud computing solutions can effectively deliver on this promise. Cloud computing services such as Amazon EC2 and Google App Engine 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 he/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 and Microsoft Windows Azure 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. Therefore, a flexible pricing model that, for example, discounts compute usage during non-peak hours seems adequate.

In our proposal 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 Flexible Provisioning and integrated load balancing of resources in the cloud 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.

II. Literature Survey

In Flexible provisioning and integrated load balancing of resources 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 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 his/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 is motivated by the following principles:

1) A simpler view of the cloud to the user: Today's cloud services vary in the abstraction presented to the user. On 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 he/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.

2) 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. Proper scheduling algorithm is the basis for cloud scheduling. In our project we need the scheduling algorithm to give different schedules and their associated price is less time without impacting the users waiting time.

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.

In this paper, we use two schedulers for scheduling the jobs and arriving at the price curve.

1. Greedy Scheduler
2. Deadline Division Scheduler

Greedy Scheduler works for the tasks which are not related. Deadline Division Scheduler works for the tasks which has dependency information.

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.

3) Benefit 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. This 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 if the shift in the price curve with respect to the price curves of the empty cloud is small. We show that Flexible provisioning and integrated load balancing of resources, gives robust price curves. Note that robust price curves imply that a cloud can meet stringent deadlines and thus satisfy more users

III. DESIGN AND IMPLEMENTATION

This chapter discusses importance of high level design its scope, description of dataflow diagrams (DFD's) function wise, for Flexible allocation and integrated load balancing of resources.

A. ***Architecture***

The system consists of two important software components Central monitor and Daemon process.

Fig.1 gives the architecture of the Central monitor which basically consists of the Graphical User Interface, where the user's selects the operations he/she wants to perform in the cloud. From Fig.1 the selected operation by the user is set as an input to the Task Schedulers which uses the algorithms such as Greedy Scheduler, Deadline Division Scheduler and Domain Clustering Scheduler. The output is generated using the Price Curve Generator which displays the results of the best case allocation of resources and how one can save the resources for execution of other tasks.

B. ***Greedy Scheduler algorithm***

The greedy algorithm always makes the choice that looks the best at that time. That is, it tries to make a locally optimal choice that could lead to the final global optimal solution in the hope. However, the greedy algorithms rarely find the globally optimal solution consistently as expected, since they usually don't operate exhaustively on all the data. The dynamic programming algorithms (or backtracking algorithms) always produce the optimal global solution, but its performance is unfavourable, especially for a large number of jobs to be scheduled Therefore, Greedy task-scheduling algorithm outstands because it is useful in real applications and it is quick to think up and often come up with good approximations to the optimum. Of course, the main benefit of greedy algorithms lies in both their conceptual simplicity and their computational efficiency. If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the actual method of choice. The combinatorial structures, known as matroids, are useful in determining when greedy method yields optimal solutions.

Let us consider a job set J which is a set of unit-time tasks with deadlines, and each J_i (1,2,…n) is independent from the other jobs, the (J, J_i) system is a matroid. Also, M=(J, J_i) is a weighted matroid with weight function w, so the Greedy-task scheduling (M, w) returns an optimal subset. By this theorem, this algorithm can be used to find a maximum weight independent job set. This method is an efficient algorithm for scheduling unit-time tasks with deadline and penalties for a single processor α=1. Its running time is O(n^2) no matter what kind of sorting algorithm applied in the process of scheduling. The strategy of the Greedy task-scheduling algorithm:

```
    Initite n time slots empty
Sort the n waiting for scheduling jobs by their weights
      For (time=1; time<=n; time++)
Schedule the highest weighted job $J_i$ (1,2,...n) in the
currently waiting queue
      If ($J_i$ deadline time slot is empty)
        then assign $J_i$ at that time slot
    else if (before $J_i$ deadline time slots available)
then assign the latest available time slot in deadline to $J_i$
Otherwise pick the most end time slot to $J_{i.}$
```

C. Deadline Division Scheduler

In order to give the user a guarantee of service, we tag every query with a deadline. This deadline refers to the latest point in time the query has to be assigned to a server for execution as soon as the deadline of a query expires. The scheduler has no other choice than assigning this very query to a server.

We tag all queries with a time stamp according to their arrival. In other words queries are not forced by deadlines to overtake others, though it is often beneficial. As a result we need to only check the first query of the current top n batch for deadline expiration. If the first query's deadline is expired we do not need to examine any other query in the batch but have to assign the first immediately to a server. Otherwise, if the first query's deadline is not expired, no other deadline can be due. Testing for expiration after the first query has been checked against all servers ensures the best assignment in case the deadline is expired. The strategy of Deadline Division algorithm is as followed,

```
while queue not empty do
        $C_{min}$ <- ∞
for each query d in $top_n$ (queue) do
for i=1 to number of servers do
        C=d ($Q,S_i$)+ W (Q)+ J($S_i$)
if C< $C_{min}$ do
        q <- Q
        s <- $S_i$
        $C_{min}$ <- C
done
done
if expired (Q) then break
done
assign query q to server s
remove q from queue
done
```

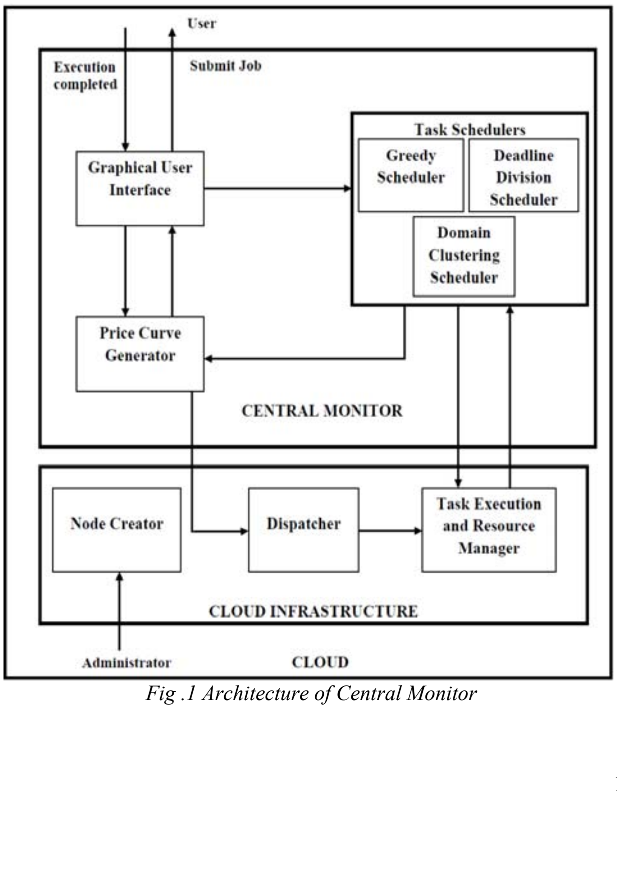
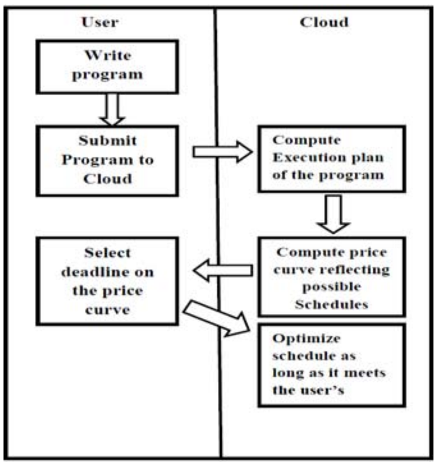


Fig .1 Architecture of Central Monitor

D. Workflow

The workflow of the system and user interaction with the system is given as shown in the Fig.2.


User
Cloud
Write program
Submit Program to Cloud
Compute Execution plan of the program
Select deadline on the price curve
Compute price curve reflecting possible Schedules
Optimize schedule as long as it meets the user's
Fig.2 Workflow

In Fig.2 the user writes a program for the execution which he/she later submits the program to the cloud for the computation of the execution plan. The cloud computes the price curve that displays all the possible schedules with respect to the user's request, which are then presented to the user. The user then selects the possible deadline on the price curve and submits it to the cloud. The selection from the user is optimized as long as it meets the user's deadline.

E. State Diagram and Sequence Diagram

The state diagram when user interacts with system is documented below,

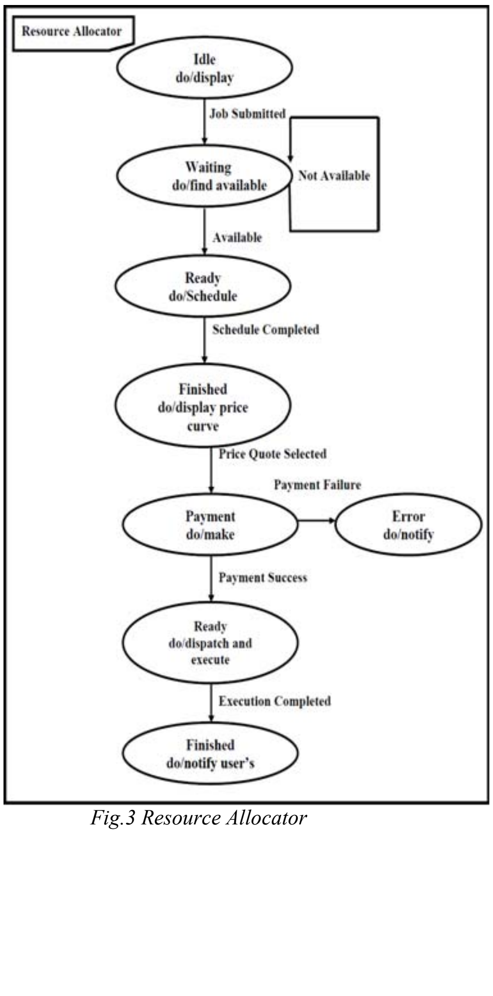


Fig.3 Resource Allocator

Initially the system state is idle where the user will be displayed with an interface and from where he/she can submit a job. Once the job is submitted the system (also can be referred to as cloud) goes into the waiting state where it finds for the availability of resources in accordance to the submitted job. If the resources are available in the system, the system goes into the ready state where it schedules the job using the task scheduling algorithms as discussed above. After the scheduling is completed it displays the price curve and sends it to the user. The user selects the price that he is intended to execute the job with and submits it to the cloud. The system prepares itself for the payment from the user and once the payment is received the particular resources selected by the user for job execution is dispatched and the job is executed. It returns back the executed job to the user with the possible outputs.
The sequence diagram for the following flow between the user and the job is shown in Fig.4, which displays all the possible interactions.

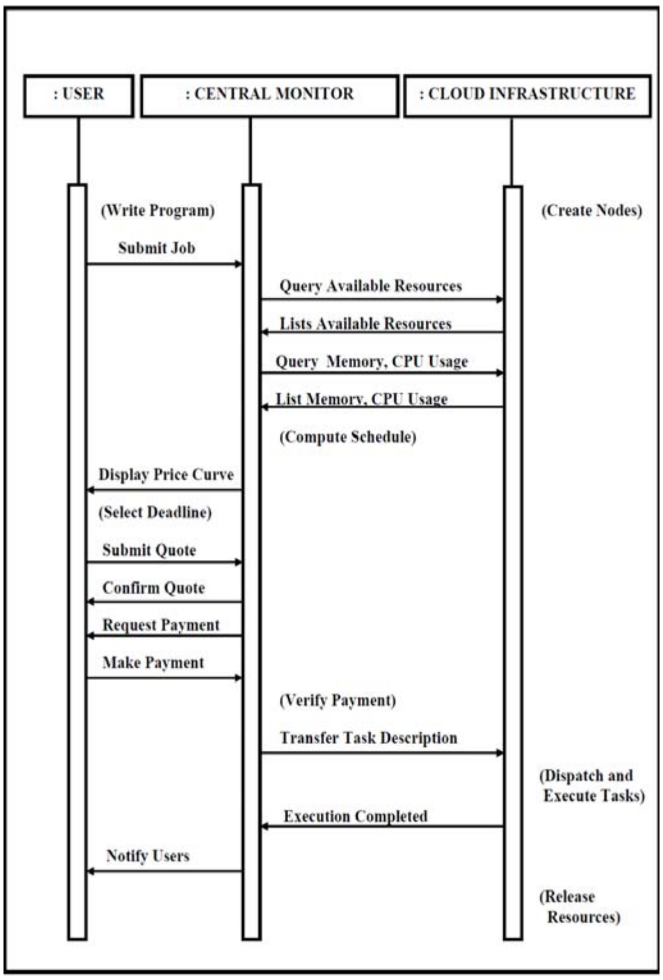


Fig.4 Sequence Diagram

IV IMPLEMENTATION RESULTS

There are two kinds of users Admin and the Cloud customer.

Admin has the following requirements.

- Admin will start the central managed cloud server
- Admin will be able to install price daemon on any node.
- Admin will register the price daemon to the process central managed server.
- Admin will define the setup price in terms of CPU, Memory and Bandwidth cost for each node.
- Admin will define the tasks which can be executed on the cloud.

Cloud User has the following requirements.

- Cloud user can submit the jobs to execute.
- The system must provide the price chart with different schedules for different execution time and cost.
- User can execute the schedule on the price chart.

The requirements can be documented in a use case diagram as shown in Fig.5.

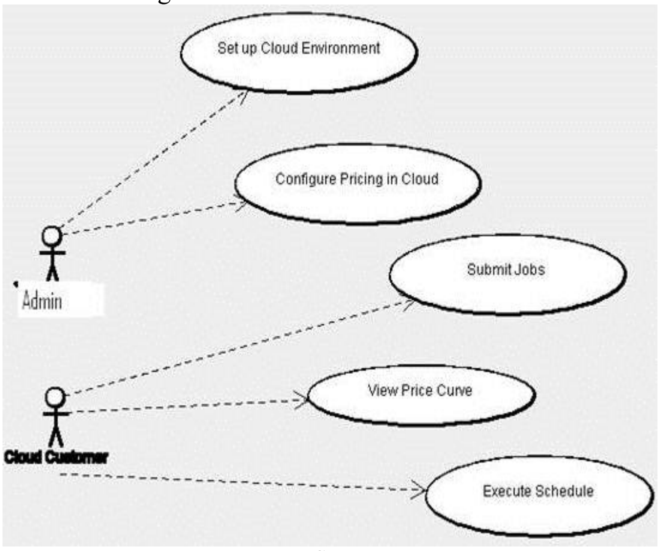
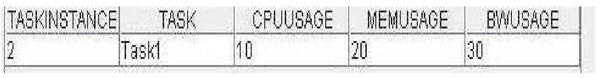


Fig. 5 Use Case Diagram

The cloud first listens to the port that the server is trying to connect with as shown in Fig.6.

Fig.6 Server listens to the port the server is trying to connect

Once the server connects it displays all the resources such as CPU usage the task is going to take the Memory usage and the Bandwidth usage, it has to the cloud user as shown in Fig. 7.



TASKINSTANCE	TASK	CPUUSAGE	MEMUSAGE	BWUSAGE
2	Task1	10	20	30

Fig. 7 Task Window

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. price to available resources as shown in Fig. 8. The user then chooses the needed resources to execute the task where each resource is assigned a price. Once the user selects the amount of resources that he/she needs to execute and pays the price for those resources the cloud releases those resources and assigns them to that particular port as shown in Fig. 9.

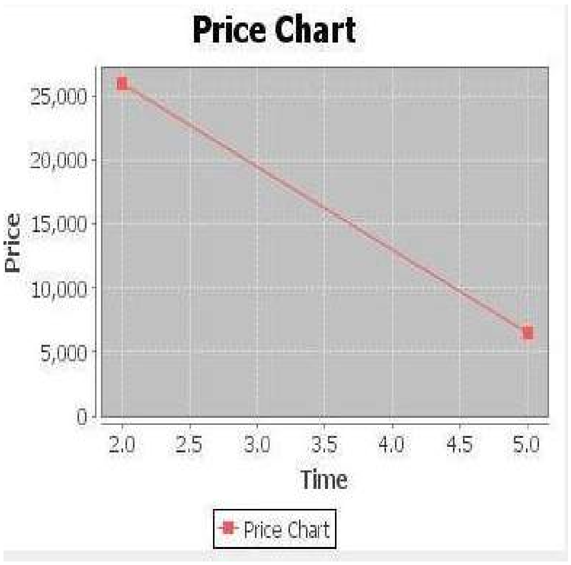
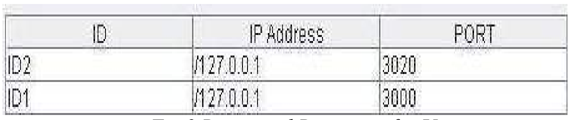


Fig.8 Price Chart



ID	IP Address	PORT
ID2	/127.0.0.1	3020
ID1	/127.0.0.1	3000

Fig.9 Registered Resources for Users

The experiments are conducted on Windows platform with programming language as JAVA.

IV. Conclusions

Scheduling is fundamental to the achievement of high performance in parallel and distributed systems. The classic work on multiprocessor scheduling dates back to 1977, 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.

Various programming models for data oriented and computation oriented programs have also been developed. Data oriented programming models include the MapReduce framework. These models minimize the transfer of data, and maximize the parallel execution of independent tasks. The MapReduce model has been widely studied.

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, Google App Engine, Microsoft Azure, and Force.com are few popular services. Cloud era 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. All these efforts study datacenter management as a programming problem and related issues.

This project 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 Flexible Provisioning and Integrated Load Balancing of Resources which achieves this transparency. We have also designed and implemented 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 project to study the robustness of price curves in a scenario where a cloud executes many simultaneous jobs.

Acknowledgment

This paper would not have existed without my guide Mrs. Poornima A B. I also would like to thank our head of the department Dr. D. R. Ramesh Babu, Prof. Bhaskar Rao, Prof. Manjunath S. S and my colleague Ajay Kumar V.

References

[1] Internet Assigned Numbers Authority. (2011, July) IANA Internet Protocol Port Numbers. http://www.iana.org/assignments/port-numbers.
[2] Wikipedia, 'Cloud Computing' available at http://en.wikipedia.org/wiki/Cloud_computing.
[3] "Enterprise Cloud Computing", http://salesforce.com/platform.
[4] "Cloudera", http://www.cloudera.com/.
[5] "Apache Hadoop", http://wiki.apache.org/hadoop.
[6] "Amazon Elastic ompute Cloud", http://aws.amazon.com/ec2.
[7] "Amazon EC2 Spot Instances", http://aws.amazon.com/ec2/spot-instances.
[8] 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.
[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, pp. 107–113, 2008.