A Collection of Three Recent Studies on Cloud Computing

Cloud Computing and Distributed Systems (CLOUDS) Laboratory,
School of Computing and Information Systems
The University of Melbourne, Australia

Adel Nadjaran Toosi
anadjaran@unimelb.edu.au
http://adelnadjarantoosi.info
Outline

- Introduction
- An Auction Mechanism for a Cloud Spot Market
  - Spot instance pricing as a Service
- Capacity Control in Infrastructure as a Service Cloud Markets
- Geographical Load Balancing for Sustainable Cloud Data Centers
- Current Research
- Summary
Outline

- Introduction
- An Auction Mechanism for a Cloud Spot Market
  - Spot instance pricing as a Service
- Capacity Control in Infrastructure as a Service Cloud Markets
- Geographical Load Balancing for Sustainable Cloud Data Centers
- Current Research
- Summary
University of Melbourne (Parkville Campus)
Doug McDonell Building (CIS)
Cloud Computing

- Allowing businesses to outsource their IT facilities to cloud providers
- Avoid expensive up-front investments of establishing their own infrastructure
- Long-held dream of computing as a utility
- On-demand delivery of IT services
- Customers pay for what they use
- Virtualized resources
Cloud Service Stack

SaaS (Software as a Service)

PaaS (Platform as a Service)

IaaS (Infrastructure as a Service)

Resources (Compute, Storage, Network)
Gartner Hype Cycle for Emerging Technologies, 2017

As of July 2017

gartner.com/SmarterWithGartner

Source: Gartner (July 2017)
© 2017 Gartner, Inc. and/or its affiliates. All rights reserved.
Outline

- Introduction
- An Auction Mechanism for a Cloud Spot Market
  - Spot instance pricing as a Service
- Geographical Load Balancing of Web Applications for Sustainable Data Centers
- Capacity Control in Infrastructure as a Service Cloud Markets
- Summary
Dynamic Pricing and Auction

- Computational resources sold by a cloud provider can be characterized:
  - As a non-storable or perishable commodity

- Demand for computational resources is:
  - non-uniform over time

- These motivates the use of dynamic forms of pricing in order to optimize revenue

- Well-designed auction mechanisms are particularly effective
Background

- **Auction:**
  - A common market mechanism with a set of rules determining prices and resource allocations on basis of bids submitted from the market participants.

- **Goals in designing auction:**
  - Truthfulness,
  - Revenue maximization,
  - Allocative efficiency,
  - Fairness
Background (Cont.)

- **Optimal mechanism design**
  - When goal of mechanism is to maximize the profit or revenue for the seller (provider)

- **Optimal mechanism design can be categorized in**
  - **Bayesian optimal mechanism design**
    » the valuations of the participants in the auction are drawn from a known prior distribution
    » based on the seminal work by Myerson
  - **Prior free optimal mechanism**
    » Determining the prior distribution is not practical, convenient or even possible in advance
    » Digital Goods: Competitive Framework by Goldberg and Hartline
Auction Mechanisms

- Amazon Web Services (AWS)
  - Spot Instances (Auction-like mechanism)
    » Customers communicate their bids for an instance hour to AWS.
    » AWS reports a market-wide spot price at which VM instance use is charged,
    » while terminating any instances that are executing under a bid price lower than
      the market price.

- The design of an auction mechanism is a Open research challenge and of great interest to cloud providers.
Spot Market and Auction

Bids

Spot Market

Cloud Provider
Our Auction Model

- Aimed at maximizing profit (generates near optimal profit for the provider)
- A multi-unit, online recurrent auction, two-dimensional bid domain
- Single Price
- A priori free
- Envy-free
- Truthful with high probability

Optimal Auction Design

- Optimal Single Price Auction: $\mathcal{F}(d)$

- The order-independent auction is truthful.

- Optimal order-independent: $\mathcal{F}(d_{-i})$
  - Not single price (i.e., not envy-free)
  - It is not fair

- How to compute a single price for an order-independent auction while attaining the revenue of the optimal auction?
Consensus Revenue Estimate

- We are interested in a mechanism that provides us with a sufficiently accurate estimate of $\mathcal{F}(d)$.
- While the estimate is constant on $d_{-i}$ for all $i$ (i.e., it achieves consensus).

$\mathcal{F}(d_{-i})$ is limited by a constant fraction of $\mathcal{F}(d)$.

It is possible to pick a good estimate:
- We consider a limit on maximum quantity ($r$)
- Let $r$ denote the supremum of the number of requested units in $d$
- Let $m$ be the number of sold units in $F$

$$\frac{m - r}{m} \mathcal{F}(d) \leq \mathcal{F}(d_{-i}) \leq \mathcal{F}(d)$$

- We use Consensus Revenue Estimate
Reserve Price

- **The reserve price** for most perishable goods and services is considerably low at their expiration time
  - A reserve price is the lowest possible price that the provider accepts.
  - The reserve price for flight seats is theoretically negligible.
- A significant part of the service cost in cloud data centers is related to power consumption of physical servers.
- We propose a method for calculating *dynamic reserve prices* based on a cost model that incorporates data center PUE (load, outside temperature), and electricity cost.
The Online Ex-CORE Auction

Algorithm 3 The Online Ex-CORE Auction

Input: \( d, p_{\text{cur}}, p_{\text{optprev}} \) \( d \) is the list of orders, sorted in descending order of bids, \( p_{\text{cur}} \) is current market price, \( p_{\text{optprev}} \) is the optimal single price in the previous round.

Output: \( p \) \( p \) Sale Price

1: \( p_{\text{opt}} \leftarrow \text{opt}(d) \)
2: if \( p_{\text{opt}} = p_{\text{optprev}} \) then
3: \( \text{return } p_{\text{cur}} \)
4: end if
5: \( r \leftarrow \text{the largest } r_i \text{ in } d \)
6: \( m \leftarrow \arg\max_{i} b_i \sigma_i(d) \)
7: if \( m \leq r \) then
8: \( \text{return } p_{\text{opt}} \)
9: else
10: \( \rho \leftarrow \frac{m}{m-r} \)
11: Find \( c \) in \( \rho \ln(c) + \rho - c = 0 \)
12: \( u \leftarrow \text{rand}(0, 1) \) \( u \) chosen uniformly random on \([0, 1]\)
13: \( l \leftarrow \lfloor \log_{e}(F(d)) - u \rfloor \)
14: \( R \leftarrow e^{l+u} \)
15: \( j \leftarrow \text{the largest } k \text{ such that } \frac{R}{\sigma_i(d)} \geq b_k \)
16: \( \text{return } \frac{R}{\sigma_j(d)} \)
17: end if
Benchmark Algorithms

- Optimal Single Price Auction (OPT)
- Holding Time Aware Optimal Auction (HTA-OPT)
- Uniform Price Auction
Performance Evaluation (Single Round)

Ratio of gained revenue by the Ex-CORE auction to optimal auction under different distribution of orders
Performance Evaluation (Online)

Average profit gained and with different auction mechanisms.
Results

Reserve price (green dashed line) and spot market price generated by online Ex-CORE (blue solid line) in a sample simulation run when the number of orders is 1500.
Spot instance pricing as a Service

- **The implementation of the proposed auction mechanism**
  - by identifying a framework called Spot instance pricing as a Service (SipaaS).

- **SipaaS:**
  - is an open source project offering a set of web services to price and sell VM instances in a spot market.

- **Utilizing SipaaS**
  - Cloud providers require installation of add-ons in their existing platform to make use of the framework.
  - An extension to the Horizon – the OpenStack dashboard project
    - To employ SipaaS web services and to add a spot market environment to OpenStack.

System Model

The diagram illustrates the components of a system model within a cloud computing environment. It includes:

- **SipaaS Framework**
- **Database**
- **Pricing Module**
- **Web Services**
- **User Interface**
- **Cloud Customers**
- **Extension Module**
- **Response**
- **Cloud Provider**
- **Resource Manager**

The **Web Services Call** from the **SipaaS Framework** to the **Extension Module** is depicted, illustrating the flow of interaction within the system model.
SipaaS Architecture

Web Services

Dispatcher Servlet
(Front Controller)

Controller
(Java Class)

View
(JSON Objects)

Pricing Module
(Java Class)

Hibernate
(Java Library)

Relational Database

Http Request
(REST Call)

Http Response
(JSON objects)
List web services

<table>
<thead>
<tr>
<th>Web Service Name</th>
<th>Input parameter(s)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>name</td>
<td>credentials</td>
</tr>
<tr>
<td>unregister</td>
<td>accesskey, secretkey</td>
<td>status</td>
</tr>
<tr>
<td>regvmttype</td>
<td>accesskey, secretkey, type</td>
<td>status</td>
</tr>
<tr>
<td>unregvmttype</td>
<td>accesskey, secretkey, type</td>
<td>status</td>
</tr>
<tr>
<td>setavailables</td>
<td>accesskey, secretkey, vmttype, quantity</td>
<td>price</td>
</tr>
<tr>
<td>setmaxqty</td>
<td>accesskey, secretkey, vmttype, quantity</td>
<td>status</td>
</tr>
<tr>
<td>setreserveprice</td>
<td>accesskey, secretkey, vmttype, value</td>
<td>price</td>
</tr>
<tr>
<td>setmaxprice</td>
<td>accesskey, secretkey, vmttype, value</td>
<td>status</td>
</tr>
<tr>
<td>addorder</td>
<td>accesskey, secretkey, vmttype, quantity, bid, ref</td>
<td>price</td>
</tr>
<tr>
<td>updateorder</td>
<td>accesskey, secretkey, quantity, ref</td>
<td>price</td>
</tr>
<tr>
<td>removeorder</td>
<td>accesskey, secretkey, ref</td>
<td>price</td>
</tr>
<tr>
<td>pricehistory</td>
<td>accesskey, secretkey, vmttype, fromdate, todate</td>
<td>price(s)</td>
</tr>
</tbody>
</table>
Extensions for Horizon

- To add spot market facilities to OpenStack,
  - Extended Horizon to be capable of using the services provided by SipaaS.

- We added a new panel through which system administrators are capable of enabling spot market support
  - Maximum and minimum amount of bid price for users,
  - Number of available VMs for allocation,
  - Maximum number of VMs a user can request.

- We added panels for requesting spot instances and viewing spot market price history
### Requesting spot instances web page

#### Details
- **Availability Zone:** nova
- **Instance Name:** test
- **Flavor:** m1.nano
- **Instance Count:** 3
- **Bid Amount:** 0.0190
- **Instance Boot Source:** Boot from image
- **Image Name:** cirros-0.3.2-x86_64-uec (24.0 MB)

#### Flavor Details
- **Name:** m1.nano
- **VCPUs:** 1
- **Root Disk:** 0 GB
- **Ephemeral Disk:** 0 GB
- **Total Disk:** 0 GB
- **RAM:** 64 MB

#### Project Limits
- **Number of Instances:** 0 of 20 Used
- **Number of VCPUs:** 0 of 20 Used
- **Total RAM:** 0 of 51,200 MB Used

[Submit] [Cancel]
Spot Pricing History Webpage

![Spot Pricing History Graph](image-url)
Sequence diagram of an order submission handling
Evaluation and Validation

- We conducted 20-minute experiment with 10 participants (i.e., spot market users).
  - Participants are selected from a group of professional cloud users who have satisfactory level of knowledge about the spot market.

- Participants are divided into two groups of five:
  - (i) Group T or truthful bidders and
  - (ii) Group C or counterpart bidders who have the freedom to misreport their true private values to maximize their utility

<table>
<thead>
<tr>
<th>User</th>
<th>Price Value ($)</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1, C1</td>
<td>0.0691</td>
<td>2</td>
</tr>
<tr>
<td>T2, C2</td>
<td>0.0092</td>
<td>1</td>
</tr>
<tr>
<td>T3, C3</td>
<td>0.0475</td>
<td>1</td>
</tr>
<tr>
<td>T4, C4</td>
<td>0.0232</td>
<td>2</td>
</tr>
<tr>
<td>T5, C5</td>
<td>0.0184</td>
<td>1</td>
</tr>
</tbody>
</table>
Utility Function

- Utility function for one time slot instance usage, formulated as below:

\[ u(r, b) = \begin{cases} 
(qv - rp)x, & \text{if } b \geq p \text{ and } r \geq q; \\
0, & \text{otherwise.} 
\end{cases} \]

- \( r \): requested number of instances
- \( b \): bid price value
- \( q \): true private quantity
- \( v \): true private price value
- \( p \): spot market price at the time of order submission
- \( x \): A Boolean value describing whether the order is accepted or not, respectively.
Results

Spot market price fluctuation during the experiment.
Results (Cont.)

<table>
<thead>
<tr>
<th>User</th>
<th>Total Cost ($)</th>
<th>Number of Full Time Slots</th>
<th>Utility Value ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1.2964</td>
<td>16</td>
<td>0.9148</td>
</tr>
<tr>
<td></td>
<td>1.8216</td>
<td>17</td>
<td>0.5278</td>
</tr>
<tr>
<td>T2</td>
<td>0.0000</td>
<td>0</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>10.0227</td>
<td>18</td>
<td>-9.8571</td>
</tr>
<tr>
<td>T3</td>
<td>0.1896</td>
<td>6</td>
<td>0.0954</td>
</tr>
<tr>
<td></td>
<td>0.2280</td>
<td>8</td>
<td>0.1520</td>
</tr>
<tr>
<td>T4</td>
<td>0.0436</td>
<td>1</td>
<td>0.0030</td>
</tr>
<tr>
<td></td>
<td>3.6810</td>
<td>5</td>
<td>-3.4490</td>
</tr>
<tr>
<td>T5</td>
<td>0.0000</td>
<td>0</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>0.0738</td>
<td>2</td>
<td>-0.0370</td>
</tr>
</tbody>
</table>

Total cost, the number of full time slots usage, and utility value of experiment participants.
Scalability of the framework

- Response time is the main performance metric taken into account in the design and implementation of service-oriented systems.
- We evaluated the scalability of the system in terms of response when demand (i.e., number of orders) grows.
- To evaluate the responsiveness of the framework,
  - We developed a web robot (Bot) application that generates order requests and submits them to the framework.
  - It generates RESTful requests based on Application Programming Interfaces (APIs) of SipaaS and measures the response time delay for each request.
Order Generation

- **Bid prices:** bid prices are drawn from a uniform distribution bounded by (0; $0.4500].
- **Quantity:** The value for the requested number of instances in each order is drawn from a uniform distribution bounded by [1, 20]
- **Arrival time:** The arrival time of the order requests are generated following a Poisson process
- **Holding time:** is modeled by Pareto distributed random variables
Testbed

- The testbed used to deploy SipaaS web services for performance evaluation:
  - HP EliteDesk 800 machine with following hardware specifications:
  - Intel(R) Core(TM) i7 Processors @ 3.6GHz.
  - 16 GB, 1600 MHz DDR3 SDRAM.
  - Seagate ST500LM000 HPDA WU 500GB.
Figure 10. The mean and 95% Confidence Interval (CI) of the response time by the SipaaS framework against the number of submitted orders.
Outline

- Introduction
- An Auction Mechanism for a Cloud Spot Market
  - Spot instance pricing as a Service
- Capacity Control in Infrastructure as a Service Cloud Markets
- Geographical Load Balancing for Sustainable Cloud Data Centers
- Current Research
- Summary
IaaS providers’ various pricing plans (or markets)

• on-demand pay-as-you-go
  » Generates highest revenue per unit capacity
  » Demand uncertainty

• Reservation (subscription)
  » Risk-free income from reservations
  » Guaranteed cash flow through long-term commitments
  » Compensate for the demand uncertainty of on-demand pay-as-you-go
  » Generates lower revenue per unit capacity
  » The provider is liable to offer guaranteed availability to honor SLA

• Spot market
  » Selling spare capacity in the data center
  » Attract price-sensitive users that are capable of tolerating service interruptions
  » Without being exposed to the risks resulting from overbooking capacity
Problem definition

The use of multiple pricing plans introduces:

- Non-trivial trade-offs in revenue maximization

Our main research question is the following:

“with limited resources available, and considering the dynamic and stochastic nature of customers’ demand, how can expected revenue be maximized through an optimal allocation of capacity to each pricing plan?”
Our Solution

- **We frame our algorithmic contributions**
  - Within a revenue management framework
  - Supports the three presented pricing plans
  - Incorporates an admission control system for requests of the reservation plan.

- **We formulate the optimal capacity control problem**
  - As a finite horizon Markov Decision Process (MDP)
  - A stochastic dynamic programming technique
    » To compute the maximum number of reservation contracts the provider must accept
    » For a large capacity provider the stochastic dynamic programming technique is computationally prohibitive.
  - We therefore present two algorithms to increase the scalability of our solution
Contributions

- **Proposed Algorithms**
  - **Optimal Algorithm**
    - A stochastic dynamic programming technique
    - As a finite horizon Markov Decision Process (MDP).
  - **Pseudo Optimal Algorithm**
    - Based on optimal algorithm
    - Only increases the spatial and temporal granularity of the problem
    - To solve it in a time suitable for practical online decision making
  - **Heuristic Algorithm**
    - Sacrifices accuracy to an acceptable extent to increase scalability
    - Through a number of simplifying assumptions on
      - Reserved capacity utilization and
      - The lifetime of on-demand requests.
Pseudo Optimal

**Details of Optimal Algorithm**

- **Computational Complexity**
  \[ O(\tau \times C^5 \times |U|^2) \]

- **B**: The number of VM instances per block of capacity
  - e.g., \( B = 100 \) VMs

- **T**: The number of billing cycles per time slot
  - e.g., \( T = 168 \) hours.

---

**Algorithm 1 Pseudo Optimal Algorithm**

```
Input: \( t, l^t, f^t, i_t \)
Output: maxrev
1: \[ dp \leftarrow \{-1\} \] \( \triangleright \) matrix \( dp \) is used for memoization and all cells are initialized with -1.
2: function \( V(t, l^t, f^t, i_t) \)
3: if \( dp[t][l^t][f^t][i_t] \neq -1 \) then
4: return \( dp[t][l^t][f^t][i_t] \)
5: end if
6: if \( t = \tau \) then
7: \( dp[t][l^t][f^t][i_t] = 0 \)
8: return 0
9: end if
10: maxrev \leftarrow 0
11: for \( r_t \leftarrow 0 \) to \( \min(C - l^t_t, f^t_t, d^t_t) \) do
12: \( \text{rev} \leftarrow 0 \)
13: \( l^t_{t+1} \leftarrow l^t_t + r_t - \sigma \frac{t^t}{\tau} \)
14: \( \alpha_t \leftarrow \min(C - l^t_t - l^t_{t+1} - r_t, d^t_t) \)
15: \( s_t \leftarrow \min(C - (l^t_t + r_t)u_t - l^t_t - \alpha_t, d^t_t) \)
16: \( \lambda_t \leftarrow (t^t - t)/\tau \)
17: \( \gamma(\zeta_t, r_t) \leftarrow B\lambda_t r_t \phi + BT(\alpha p(l^t_t + r_t)u_t + p(l^t_{t+1} + \alpha_t) + \beta ps_t) \)
18: for \( l^t_{t+1} \leftarrow 0 \) to \( \alpha_t \) do
19: for \( i_{t+1} \leftarrow 0 \) to \( |U| \) do
20: \( p(\xi_{t+1}|\zeta_t, r_t) \leftarrow \text{Bin}(l^t_{t+1}, l^t_t + \alpha_t, q) \times p(u_t = u_{t+1}) \)
21: \( \text{rev} \leftarrow \text{rev} + \gamma(\zeta_t, r_t) + p(\xi_{t+1}|\zeta_t, r_t) \times V(t + 1, l^t_{t+1}, f^t_{t+1}, i_{t+1}) \)
22: end for
23: end for
24: if \( \text{rev} \geq \text{maxrev} \) then
25: \( \text{maxrev} \leftarrow \text{rev} \)
26: end if
27: dp[t][l^t][f^t][i_t] \leftarrow \text{maxrev}
28: return maxrev
29: end function
```
The heuristic algorithm

- Computational Complexity: $O(\tau \times C)$
Key modules of the revenue management framework

Requests

History

Collector

Reserved Capacity Analyzer

Prediction Module

Future Demands (d)

Utilization (u)

Admission Controller

Algorithm

Accept

Reject

Revenue Management Framework
Performance Evaluation

- We evaluate our proposed framework through:
  - Large-scale simulations (12 months)
  - Driven by cluster-usage traces that are provided by Google.

- No publicly available workload traces of real-world IaaS clouds:
  - We propose a scheduling algorithm that generates VM requests based on the user resource usage in these traces.

- Pricing conditions that are aligned with those of Amazon EC2

- Benchmark Algorithm:
  - no admission control referred to as *no-control*

- Simulation Environment
  - Extended CloudSim (pricing plans and the proposed revenue management framework)
Experimental Results (1)

The revenue performance of the proposed revenue management framework under different algorithms normalized to the outcome of no-control algorithm ($B = 100$ and $T = 75$).
Experimental Results (2)

The revenue performance of the pseudo optimal and heuristic algorithms with different values of B and T. All values are normalized to the outcome of the optimal solution.
Outline

- Introduction
- An Auction Mechanism for a Cloud Spot Market
  - Spot instance pricing as a Service
- Capacity Control in Infrastructure as a Service Cloud Markets
- Geographical Load Balancing for Sustainable Cloud Data Centers
- Current Research
- Summary
Large Energy Consumption

- Data centers used for hosting cloud applications consume large amount of electricity
  - High operational cost for the cloud providers
  - High carbon footprint on the environment.

  - US data centers alone consumed 91 billion kilowatt-hours of electricity, equivalent to two-year power consumption of all households in New York city
  - This is projected to be responsible for the emission of nearly 50 million tons of carbon pollution per annum in 2020.
Renewable Energy

- **Cloud service providers are working hard:**
  - To reduce their energy consumption
  - Their dependence on power generated from fossil fuels (i.e., Brown energy)

- **Using renewable energy**
  - Direct investments in on-site green power generation
    - Companies such as Google, Microsoft and Amazon

  “Amazon Web Services (AWS) is building a wind farm that will be operational by end-2016 that generates 40 percent of its electrical usage from renewable energy sources”
Challenge

- Intermittency and unpredictability of renewable energy sources
  - Powering data centers entirely with renewable energy sources is difficult.

- Sources of energy for data centers:
  - Grid power or brown energy
  - Renewable energy sources or green energy

- Challenge:
  - To minimize brown energy usage
  - To maximize renewable energy utilization
Geographical Load Balancing (GLB)

“with limited or even no a priori knowledge of the future workload, and dynamic and unpredictable nature of renewable energy sources, how does one allocate requests to each data center such that the total cost of power consumption is minimized and the overall renewable energy utilization is maximized?”
Offline GLB Problem
Time Complexity

- This leads to the $O(n^m)$ different state variables.
- **Exponential time complexity** given a bounded number of data centers.
- The complexity can be improved using techniques such as branch and bound.
  - It does not reduce the worst-case time complexity.
We present a fuzzy logic-based load balancing algorithm that only uses recent data history to tackle the geographical load balancing problem.

Why fuzzy logic?

- Fuzzy logic is conceptually easy to understand and mathematical concepts behind fuzzy reasoning, even though subtle, are very simple.
- Fuzzy logic systems are good at dealing with uncertainty and lack of perfect information.
- Fuzzy logic reasoning is among the best techniques, for solving non-linear systems with an arbitrary complexity and large number of inputs.

Fuzzy Logic-based Load Balancing

- Input values computed within a window of size $W$ in the recent history for Datacenter $i$:
  - **The utilization of renewable energy sources ($U_i$):** is a value in the range $[0, 1]$ and is computed as the ratio of the number of used to the total number of available renewable power units.
  - **Amount of brown energy consumption ($B_i$):** is a value in the range $[0, +\infty)$ and is computed as the ratio of the total number of units of brown power units used to the total number of available renewable power units.
  - **Average price of electricity in the location ($F_i$):** is the average price for an unit of power.

- Output (Varies between 0 and 1)
  - Specifies the suitability of each data center for routing the request, where 0 shows the lowest suitability and 1 shows the highest suitability.
Membership Functions

(a) Input MFs for the Utilization of Renewable Energy ($U_i$)

(b) Input MFs for the Brown Energy Consumption ($B_i$)

(c) Input MFs for the Price of Electricity ($F_i$)

(d) Output MFs for Suitability
### Fuzzy Rules

<table>
<thead>
<tr>
<th>$U_i$</th>
<th>$B_i$</th>
<th>$F_i$</th>
<th>Suitability</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>low</td>
<td>-</td>
<td>veryhigh</td>
</tr>
<tr>
<td>low</td>
<td>high</td>
<td>-</td>
<td>high</td>
</tr>
<tr>
<td>mid</td>
<td>low</td>
<td>-</td>
<td>high</td>
</tr>
<tr>
<td>mid</td>
<td>high</td>
<td>-</td>
<td>mid</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
<td>low</td>
<td>mid</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
<td>mid</td>
<td>mid</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>low</td>
<td>mid</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>mid</td>
<td>low</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>high</td>
<td>verylow</td>
</tr>
</tbody>
</table>
Performance Evaluation

- We conduct experiments based on simulation to study the performance of fuzzy logic-based load balancing (FLB).
  - CloudSim, a discrete-event Cloud simulator.
- Our aim is to understand the renewable energy utilization and cost performance of FLB in realistic settings.
- In order to achieve our goal, we consider a case study based on real-world traces for the workload, renewable availability, and electricity prices.
Workload Setup

- **Traces of Google cluster-usage**
  - Google cluster of 12K physical servers
  - 933 users
  - Over a time period of 29 days
  - Devised Scheduling algorithm
Configuration of data centers

A map showing the locations of various data centers, including SP-15, Palo Verde, and Phoenix.
Renewables and Electricity prices

- **Renewable Energy**
  - Meteorological data traces database of National Renewable Energy Laboratory (NREL)
  - Wind turbines (GE 1.5MW wind turbine with efficiency of 40%) and solar panels (500m^2 with efficiency of 30%)
  - The model proposed by Fripp and Wiser is employed where the wind speed, the air temperature, and the air pressure measurements in the location of each data center fed into the model.

- **Electricity prices**
  - Wholesale electricity price information for each hub is collected from the website of Energy Information Administration (EIA)
Experimental Setup (Renewables and Electricity prices)

Wholesale electricity market price for the hubs

Renewable power generation for five days.
Benchmark Algorithms:

- **Future-Aware Best Fit (FABEF):** It routes each request to a data center leading to the lowest cost of the request accommodation irrespective of upcoming requests.

- **Round Robin (RR):** It routes requests to data centers in circular order with null information about the status of data centers.

- **Highest Available Renewable First (HAREF):** It routes requests to the data center with the highest amount of available renewable energy at the current time slot.
Results

The total cost of different algorithms normalized to the outcome of the RR algorithm.

The green power utilization of different algorithms normalized to the outcome of the RR algorithm.
Results

Effect of window size on the total cost performance of FLB and FABEF normalized to the outcome of the RR algorithm.
Overall system architecture
Global Load Balancer

- **HAProxy**:
  - Fetch numbers requests for each site

- **Decision Making Module**
  - HAPerxy Monitoring
  - Power Monitoring

- **Controller**
  - Set weights
  - Fetch power consumption on each site
Algorithm 1 Green Load Balancing (GreenLB) Policy

1: $R \leftarrow 0$
2: for all data centers $d$ in the list do
3:   $c \leftarrow$Fetch the data center’s energy consumption in Watt-hour within the time window
4:   $t \leftarrow$ Fetch the number of requests redirected to the site within the same time window
5:   $a \leftarrow$ Fetch currently available renewable power at the site in Watt
6:   $w \leftarrow$ Compute Watt-hour consumption per request ($c / t$)
7:   $r_d \leftarrow$ Compute the request rate (#reqs/hour) data center $d$ can accommodate using renewables ($a / w$)
8:   $R \leftarrow R + r_d$
9: end for
10: $\gamma \leftarrow$ Fetch request rate (#reqs/hour) at Global-LB
11: If $\gamma < R$ then
12:   for all data centers $d$ in the list do
13:      set weight as $r_d / R$
14:   end for
15: else
16:   Find the data center $d'$ with the cheapest price of brown energy per request.
17:   $L \leftarrow \gamma$
18:   for all data centers $d$ in the list except $d'$ do
19:      set weight as $r_d / L - r$
20:   $L \leftarrow L - r$
21: end for
22: Set the weight for $d'$ as $L / \gamma$
23: end if
24: Update HAProxy weights accordingly
Grid’5000 Testbed
Prototype System
Traces

Figure 6: The English Wikipedia workload for 19th and 20th of September 2007.

Figure 7: Electricity prices for two days.

(a) Solar  (b) Wind  (c) Normalized total

Figure 8: Renewable Power Generation for two days.
Results

(a) Lyon
(b) Reims
(c) Rennes

GreenLB  Capping  Round Robin
## Results

<table>
<thead>
<tr>
<th>Site</th>
<th>Metric</th>
<th>RR</th>
<th>Capping</th>
<th>GreenLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lyon</td>
<td>Power Consumption</td>
<td>36.2</td>
<td>42.9</td>
<td>41.2</td>
</tr>
<tr>
<td></td>
<td>(kWh)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Brown Consumption</td>
<td>13.3</td>
<td>19.0</td>
<td>16.9</td>
</tr>
<tr>
<td></td>
<td>(kWh)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost (€)</td>
<td>1.71</td>
<td>2.31</td>
<td>2.01</td>
</tr>
<tr>
<td>Reims</td>
<td>Power Consumption</td>
<td>32.5</td>
<td>32.5</td>
<td>35.4</td>
</tr>
<tr>
<td></td>
<td>(kWh)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Brown Consumption</td>
<td>3.1</td>
<td>1.1</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>(kWh)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost (€)</td>
<td>0.42</td>
<td>0.15</td>
<td>0.27</td>
</tr>
<tr>
<td>Rennes</td>
<td>Power Consumption</td>
<td>36.4</td>
<td>29.7</td>
<td>28.3</td>
</tr>
<tr>
<td></td>
<td>(kWh)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Brown Consumption</td>
<td>9.3</td>
<td>2.9</td>
<td>2.6</td>
</tr>
<tr>
<td></td>
<td>(kWh)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost (€)</td>
<td>1.23</td>
<td>0.39</td>
<td>0.35</td>
</tr>
<tr>
<td>Total</td>
<td>Power Consumption</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>(kWh)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Brown Consumption</td>
<td>25.7</td>
<td>23.0</td>
<td>21.4</td>
</tr>
<tr>
<td></td>
<td>(kWh)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost (€)</td>
<td>3.36</td>
<td>2.85</td>
<td>2.63</td>
</tr>
</tbody>
</table>
Results
Outline

- Introduction
- An Auction Mechanism for a Cloud Spot Market
  - Spot instance pricing as a Service
- Capacity Control in Infrastructure as a Service Cloud Markets
- Geographical Load Balancing for Sustainable Cloud Data Centers
- Current Research
- Summary
Software Defined Networking (SDN)
Joint Host and Network VM Consolidation

- Actual utilization (energy consumption)
- Allocated capacity
- Requested capacity

VM1, VM2, VM3, VM4

Host1, Host2, Host3, Host4

No overbooking

Switches turned off

After overbooking / consolidation

Hosts turned off
OpenStack and Raspberry Switches
Outline

• Introduction
• An Auction Mechanism for a Cloud Spot Market
  • Spot instance pricing as a Service
• Capacity Control in Infrastructure as a Service Cloud Markets
• Geographical Load Balancing for Sustainable Cloud Data Centers
• Current Research
• Summary
Summary

- Discussed envy-free auction that is truthful with high probability generates a near optimal profit for the cloud provider.
- A revenue management can efficiently allocate capacity to different markets, i.e., reservation, on-demand pay-as-you and spot markets.
- Geographical Load Balancing for efficient utilizing renewable energy.
Thank you!

Questions?