Dynamic Resource Allocation for Spot Markets in Cloud Computing Environments

Qi Zhang[1], Quanyan Zhu[2], Raouf Boutaba[1,3]

[1]David. R. Cheriton School of Computer Science
University of Waterloo

[2]Department of Electrical and Computer Engineering
University of Illinois at Urbana Champaign

[3]Division of IT Convergence Engineering
POSTECH

Introduction

- Cloud computing aims at providing compute resources like public utilities
    - Resources can be rapidly acquired and released on-demand

Introduction

- Cloud computing aims at providing compute resources like public utilities
  - Resources can be rapidly acquired and released on-demand
- Most of the cloud providers today uses fixed pricing schemes
  - Easy to implement, easy to budget cost

Introduction

- Cloud computing aims at providing compute resources like public utilities
  - Resources can be rapidly acquired and released on-demand
- Most of the cloud providers today uses fixed pricing schemes
  - Easy to implement, easy to budget cost
- However, fixed pricing scheme can unsuitable for on-demand resource allocation
  - Low demand causes low resource utilization
  - High demand may cause request rejection, resulting in low customer satisfaction

Introduction

- Cloud computing aims at providing compute resources like public utilities
  - Resources can be rapidly acquired and released on-demand
- Most of the cloud providers today uses fixed pricing schemes
  - Easy to implement, easy to budget cost
- However, fixed pricing scheme can unsuitable for on-demand resource allocation
  - Low demand causes low resource utilization
  - High demand may cause request rejection, resulting in low customer satisfaction
- Market-based resource allocation is gaining popularity
  - Let the price fluctuates with supply and demand

Amazon EC2 Spot Instance Service

- Launched on Dec. 15, 2009
- Multiple VM types per availability zone
- Customers submit requests the specify number of VMs and bidding prices
- Spot price fluctuates with supply and demand according to Amazon
- Instances may be terminated without prior notice

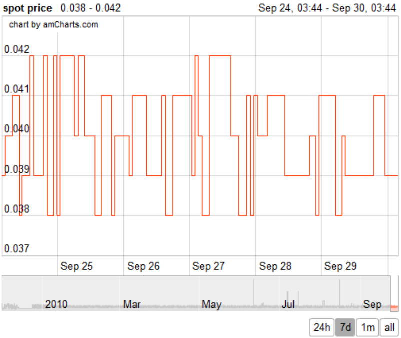


Figure 1: Price of a small Linux instance in a week

Introduction (Con't)

- Energy is another major concern of Cloud providers
  - Accounts for 20% of total annual expense

Introduction (Con't)

- Energy is another major concern of Cloud providers
  - Accounts for 20% of total annual expense
- The best way to save energy is to set unused servers to a power-saving state (e.g. turn them off)

Introduction (Con't)

- Energy is another major concern of Cloud providers
  - Accounts for 20% of total annual expense
- The best way to save energy is to set unused servers to a power-saving state (e.g. turn them off)
- However, frequently switching a server in and out of power-saving state will cause wear-and-tear effect and reduce its life time
  - It is necessary to model this penalty in the cost function

Motivation

- Multiple spot markets sharing the same data center capacity
  - As request arrival can be highly dynamic, sometimes certain markets may be hotter than others

Motivation

- Multiple spot markets sharing the same data center capacity
  - As request arrival can be highly dynamic, sometimes certain markets may be hotter than others
- The dynamic capacity provisioning problem
  - When demand is high, decide how many resources should be allocated to each market
  - When demand is low, decide how many servers should be set to the sleep state

Motivation

- Multiple spot markets sharing the same data center capacity
  - As request arrival can be highly dynamic, sometimes certain markets may be hotter than others
- The dynamic capacity provisioning problem
  - When demand is high, decide how many resources should be allocated to each market
  - When demand is low, decide how many servers should be set to the sleep state
- There are penalties for adjusting both price and capacity
  - Rapid change of prices can cause frequent preemption of customer's tasks
  - Rapid change of capacity can hurt server lifetime

Our Contribution

- We study the online dynamic capacity provisioning problem

Our Contribution

- We study the online dynamic capacity provisioning problem
- We formulate dynamic capacity provisioning as an optimization problem that considers
  - Demand fluctuation
  - Energy cost
  - Penalty for capacity adjustment

Our Contribution

- We study the online dynamic capacity provisioning problem
- We formulate dynamic capacity provisioning as an optimization problem that considers
  - Demand fluctuation
  - Energy cost
  - Penalty for capacity adjustment
- We present a Model Predictive Control (MPC) framework for the dynamic capacity provisioning problem for Amazon EC2 spot markets
  - Amazon EC2 is the only cloud provider currently offer spot instance services

Related Work

- Market-based Resource Allocation
  - Most of the existing work assumes fixed capacity
  - Does not consider electricity cost

Related Work

- Market-based Resource Allocation
  - Most of the existing work assumes fixed capacity
  - Does not consider electricity cost
- Automatic Capacity Provisioning
  - Studied under Autonomic Computing
  - Does not consider economics aspects

Related Work

- Market-based Resource Allocation
  - Most of the existing work assumes fixed capacity
  - Does not consider electricity cost
- Automatic Capacity Provisioning
  - Studied under Autonomic Computing
  - Does not consider economics aspects
- Resource Allocation in Electricity Spot Market
  - Similar problem but with single type of goods
  - Control theory is widely used in this context

System Architecture

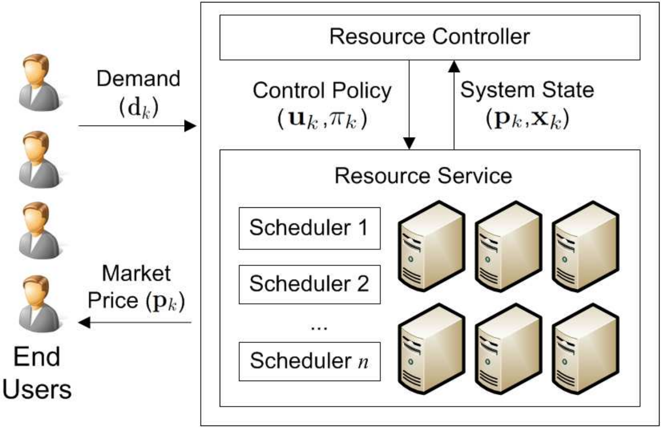


Figure 2: System Model

Demand Model

- We consider a discrete time model where time is divided into slots: $k = 1, 2, ..., \mathcal{K}$

Demand Model

- We consider a discrete time model where time is divided into slots: $k = 1, 2, ..., \mathcal{K}$
- There are N types of VMs: $i = 1, 2, ..., N$

Demand Model

- We consider a discrete time model where time is divided into slots: $k = 1, 2, ..., \mathcal{K}$
- There are N types of VMs: $i = 1, 2, ..., N$
- Let d_k^i denote the demand for VM type i at time k

Demand Model

- We consider a discrete time model where time is divided into slots: $k = 1, 2, ..., \mathcal{K}$
- There are N types of VMs: $i = 1, 2, ..., N$
- Let d_k^i denote the demand for VM type i at time k
- Let p_k^i denote the price for VM type i at time k

Demand Model

- We consider a discrete time model where time is divided into slots: $k = 1, 2, ..., \mathcal{K}$
- There are N types of VMs: $i = 1, 2, ..., N$
- Let d_k^i denote the demand for VM type i at time k
- Let p_k^i denote the price for VM type i at time k
- Demand is a monotonic decreasing function $l(\cdot)$ of price

$$d_k^i = l^i(k, p_k^i) + v_k^i \tag{1}$$

Demand Model

- We consider a discrete time model where time is divided into slots: $k = 1, 2, ..., \mathcal{K}$
- There are N types of VMs: $i = 1, 2, ..., N$
- Let d_k^i denote the demand for VM type i at time k
- Let p_k^i denote the price for VM type i at time k
- Demand is a monotonic decreasing function $l(\cdot)$ of price

$$d_k^i = l^i(k, p_k^i) + v_k^i \qquad (1)$$

- To simplify the model, we approximate $l(\cdot)$ locally using a linear function
  - This is reasonable since the model penalizes rapid price change

$$d_k^i = \bar{d}_k^i - \alpha^i(p_k^i - \bar{p}_k^i) + v_k^i \qquad (2)$$

Demand Model

- We consider a discrete time model where time is divided into slots: $k = 1, 2, ..., \mathcal{K}$
- There are N types of VMs: $i = 1, 2, ..., N$
- Let d_k^i denote the demand for VM type i at time k
- Let p_k^i denote the price for VM type i at time k
- Demand is a monotonic decreasing function $l(\cdot)$ of price

$$d_k^i = l^i(k, p_k^i) + v_k^i \qquad (1)$$

- To simplify the model, we approximate $l(\cdot)$ locally using a linear function
  - This is reasonable since the model penalizes rapid price change

$$d_k^i = \bar{d}_k^i - \alpha^i(p_k^i - \bar{p}_k^i) + v_k^i \qquad (2)$$

- We assume the model parameters can be obtained using linear regression or other methods

System Model

- We consider a data center that consists of C identical machines
  - Can be extend to multiple generations of identical machines

System Model

- We consider a data center that consists of C identical machines
  - Can be extend to multiple generations of identical machines
- For model simplicity, we assume each machine is dedicated to one VM type
  - Can be generalized to multiple VM types, assuming there are limited number of types

System Model

- We consider a data center that consists of C identical machines
  - Can be extend to multiple generations of identical machines
- For model simplicity, we assume each machine is dedicated to one VM type
  - Can be generalized to multiple VM types, assuming there are limited number of types
- Denote by x_k^i the fraction of machines dedicated to type i at time k

System Model

- We consider a data center that consists of C identical machines
  - Can be extend to multiple generations of identical machines
- For model simplicity, we assume each machine is dedicated to one VM type
  - Can be generalized to multiple VM types, assuming there are limited number of types
- Denote by x_k^i the fraction of machines dedicated to type i at time k
- Denote by u_k^i the change in the fraction of machines dedicated to type i at the end of time k

System Model

- We consider a data center that consists of C identical machines
  - Can be extend to multiple generations of identical machines
- For model simplicity, we assume each machine is dedicated to one VM type
  - Can be generalized to multiple VM types, assuming there are limited number of types
- Denote by x_k^i the fraction of machines dedicated to type i at time k
- Denote by u_k^i the change in the fraction of machines dedicated to type i at the end of time k
- Denote by e^i the energy cost of a machine dedicated to type i

System Model

- We consider a data center that consists of C identical machines
  - Can be extend to multiple generations of identical machines
- For model simplicity, we assume each machine is dedicated to one VM type
  - Can be generalized to multiple VM types, assuming there are limited number of types
- Denote by x_k^i the fraction of machines dedicated to type i at time k
- Denote by u_k^i the change in the fraction of machines dedicated to type i at the end of time k
- Denote by e^i the energy cost of a machine dedicated to type i
- State equation for capacity is

$$x_{k+1}^i = x_k^i + u_k^i \tag{3}$$

System Model (Con't)

- Denote by p_k^i the price of VM type i at time k

System Model (Con't)

- Denote by p_k^i the price of VM type i at time k
- Denote by π_k^i the change in the price of VM type i at the end of time k

System Model (Con't)

- Denote by p_k^i the price of VM type i at time k
- Denote by π_k^i the change in the price of VM type i at the end of time k
- State equation for price is

$$p_{k+1}^i = p_k^i + \pi_k^i \tag{4}$$

System Model (Con't)

- Denote by p^i_k the price of VM type i at time k
- Denote by π^i_k the change in the price of VM type i at the end of time k
- State equation for price is

$$p^i_{k+1} = p^i_k + \pi^i_k \tag{4}$$

- The spot instance service can be modeled as a $M/G/c$ queue with average arrival rate $\lambda^i_k = d^i_k/T$ and processing rate μ^i

System Model (Con't)

- Denote by p_k^i the price of VM type i at time k
- Denote by π_k^i the change in the price of VM type i at the end of time k
- State equation for price is

$$p_{k+1}^i = p_k^i + \pi_k^i \tag{4}$$

- The spot instance service can be modeled as a $M/G/c$ queue with average arrival rate $\lambda_k^i = d_k^i/T$ and processing rate μ^i
- The net income can be expressed as

$$\mathbb{E}(R_k^i) = \min\left(1, \frac{\mathbb{E}(\lambda_t^i)}{\mu^i C x_k^i}\right) p_k^i T - Ce^i x_k^i \tag{5}$$

System Model (Con't)

- Denote by p_k^i the price of VM type i at time k
- Denote by π_k^i the change in the price of VM type i at the end of time k
- State equation for price is

$$p_{k+1}^i = p_k^i + \pi_k^i \tag{4}$$

- The spot instance service can be modeled as a $M/G/c$ queue with average arrival rate $\lambda_k^i = d_k^i/T$ and processing rate μ^i
- The net income can be expressed as

$$\mathbb{E}(R_k^i) = \min\left(1, \frac{\mathbb{E}(\lambda_t^i)}{\mu^i C x_k^i}\right) p_k^i T - Ce^i x_k^i \tag{5}$$

- The net income is maximized when supply $\mu^i C x_k^i$ matches demand $\mathbb{E}(\lambda_t^i)$

System Model (Con't)

- Even though it is desirable to match supply and demand, 100% utilization can cause unacceptable queuing delay

System Model (Con't)

- Even though it is desirable to match supply and demand, 100% utilization can cause unacceptable queuing delay
- Assume there is a desirable average queuing delay, we translate it into a desirable utilization level ρ^i

System Model (Con't)

- Even though it is desirable to match supply and demand, 100% utilization can cause unacceptable queuing delay
- Assume there is a desirable average queuing delay, we translate it into a desirable utilization level ρ^i
- The objective is to minimize

$$\mathbb{E}(R) = \mathbb{E}\left[\sum_{i=1}^{N}\sum_{k=1}^{\mathbf{K}} -R_k^i + q^i(Cx_k^i - \sigma^i d_k^i)^2 + r_1^i(u_k^i)^2 + r_2^i(p_k^i)^2\right]$$

where σ^i is a constant weight factor, q^i, r_1 and r_2 are penalty factors for modeling the cost for meeting desired utilization level, changing capacity and price, respectively

Designing the MPC controller

- The optimization problem is a linear quadratic program that can be solved optimally in polynomial time

Designing the MPC controller

- The optimization problem is a linear quadratic program that can be solved optimally in polynomial time
- However, resource controller needs to solve the problem online

Designing the MPC controller

- The optimization problem is a linear quadratic program that can be solved optimally in polynomial time
- However, resource controller needs to solve the problem online
- We devise a MPC algorithm for the problem
  1. At time k, predict future demand for a window $\mathcal{K}$
  2. Solve the problem optimally to determine u_k and π_k
  3. Apply change (u_k and π_k) at the end of time slot k
  4. Repeat Step 1-3

Experiments Setup

- We have implemented the scheduler and controller in Matlab

Table 1: Types of VMs used in the experiments

VM Type	CPU Capacity (Cores)	Memory Size (MB)	average duration (seconds)	Avg. bidding price ($)
small	1	64	1694	0.038
medium	1	128	4862	0.039
large	1	256	14049	0.041

Experiments Setup

- We have implemented the scheduler and controller in Matlab
- For cloud workload, we use the publically available trace from Google compute clusters

Table 1: Types of VMs used in the experiments

VM Type	CPU Capacity (Cores)	Memory Size (MB)	average duration (seconds)	Avg. bidding price ($)
small	1	64	1694	0.038
medium	1	128	4862	0.039
large	1	256	14049	0.041

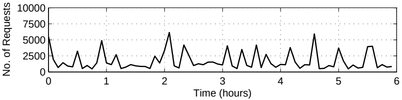
Experiments Setup

- We have implemented the scheduler and controller in Matlab
- For cloud workload, we use the publically available trace from Google compute clusters
- However, needs to pre-process the dataset
  - Match VM size with the ones used in SpotCloud
  - Generate prices from random gaussian distributions

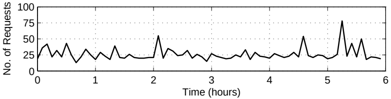
Table 1: Types of VMs used in the experiments

VM Type	CPU Capacity (Cores)	Memory Size (MB)	average duration (seconds)	Avg. bidding price ($)
small	1	64	1694	0.038
medium	1	128	4862	0.039
large	1	256	14049	0.041

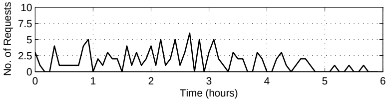
Task Arrival Rate in Google's Workload Traces



(a) Small VMs



(b) Medium VMs



(c) Large VMs

Figure 3: Task Arrival Rate in Google Workload Traces

Resource Usage and Allocation

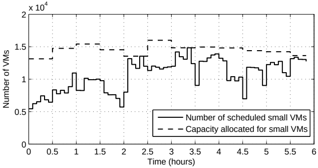


Figure 4: Num. of small VMs in the cluster

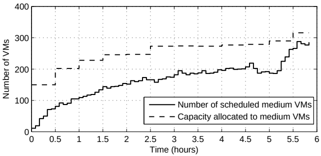


Figure 5: Num. of medium VMs in the cluster

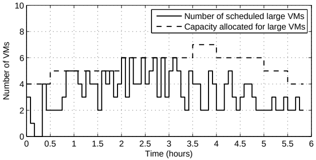


Figure 6: Num. of large VMs in the cluster

Price and Utilization

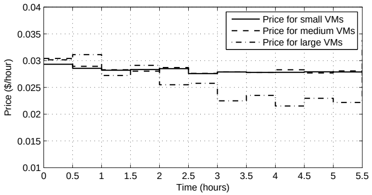


Figure 7: Price for each VM service

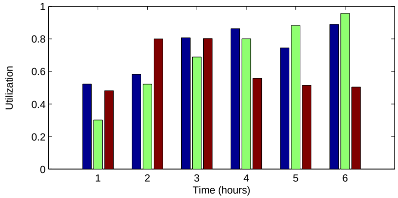


Figure 8: Utilization of allocated servers per hour

Conclusion

- Market-based resource allocation is a promising approach for resource allocation in Clouds

Conclusion

- Market-based resource allocation is a promising approach for resource allocation in Clouds
- We have presented a framework that dynamically adjust supply and price for different spot markets that considers
  - Demand fluctuation
  - Energy cost
  - Penalty for capacity adjustment

Conclusion

- Market-based resource allocation is a promising approach for resource allocation in Clouds
- We have presented a framework that dynamically adjust supply and price for different spot markets that considers
  - Demand fluctuation
  - Energy cost
  - Penalty for capacity adjustment
- Future work
  - Analyze the problem from customers point of view
  - Design incentive compatible auction mechanism that achieves optimal revenue