On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud

Ishai Menache (MSR)
Ohad Shamir (Weizmann)
Navendu Jain (MSR)

ICAC’2014
Background

• Cloud is a growing business
• More purchasing options, more variety
• Which/when/where resource should I rent??
• Pricing calculators, auto-scale mechanisms exist, but *not enough*...
• Need to *automatically* adjust purchasing decisions as a function of *dynamically* evolving conditions/workloads.
Background

• This work: *Automated* resource allocation for batch jobs
  – Focus on compute instances
  – Available options:
    • On-demand
    • Spot
    • Reserved (not in this work)
The basic tradeoff

- On demand: guaranteed, but expensive
- Spot: usually cheaper, but interruptions/delays

EC2 case studies
- Spot instances are often used
- However no principled mechanisms to choose between on-demand and spot instances.
Outline

• The model (jobs, allocations)
• The online-learning algorithm
• Experiments
• Conclusion
The job model

- Arrival (A)
- Job size (z) [instance hr]
- Parallelism constraint (c)
- Value function $V(\tau)$
  - E.g., strict deadline (d)
- Utility: $V(\tau) - \text{Cost(resources)}$

**Objective:** Maximize job utilities
Job resource allocation

- For simplicity, we restrict attention to single size
- Decisions per job: # on-demand, [# spot, bid]
  - Can modify decisions every hour
- Allocation/payment
  - On-demand:
    - Fixed price per-instance-hr
  - Spot:
    - Varying price (e.g., 5 min resolution)
    - Get instances (and pay) only if bid above market price
      - Pay the market price
Algorithm in a nutshell

• A set of parameterized policies
• “Attach” a policy to each arriving job
• Policy picked at random
  – Successful policies have higher probability of being chosen
  – Probabilities updated after each job departure
• Online-learning algorithm is essentially about how to update the probabilities.
  – Ensures good performance even though we don't know in advance which policies will work well
Parameterized policies

1. Choosing between on-demand and spot
   a. Deadline-centric: start with spot, switch to on-demand when $M$ hours from deadline
   b. Rate-centric: Fixed rate $\sigma \in [0,1]$ of on-demand instances

2. Bidding on spot instances
   a. Fixed bid $b$
   b. Variable bid $\int_y p_s(y) y^{\tau - y} dy + \epsilon$

3. Abandon job? (Value vs. cost, deadline vs. remaining compute, etc.)
   • Policy: deadline/rate centric + Fixed/variable bid
Online-learning algorithm

• Main loop: for each job j
  – For each policy $\pi$
    • Calculate $U_j(\pi)$
    • Set $w_\pi := w_\pi e^{\eta_j U_j(\pi)}$

• Note: $U_j(\pi)$ cannot be evaluated immediately
  – “delayed feedback”

• Delayed feedback not standard in online-learning
  – required developing a tailored algorithm
Algorithm guarantees

• Criterion: Regret
  \[ \max_{\pi} \frac{1}{J} \sum_{j} U_j(\pi) - \frac{1}{J} \sum_{j} U_j(\pi_j) \]

• Theorem: Regret \textbf{vanishes to zero} as total number of jobs (J) increases
  \[ - \text{Regret proportional to } 1/\sqrt{J} \]
Evaluation

• Simplified assumptions:
  – No checkpointing overhead
  – No delays in obtaining requested instances

• Synthetic data
  – Facilitates “debugging” the algorithm

• Real data
  – EC2 spot-price history
  – Map-reduce job traces
Simulations - synthetic data

• Setup:
  – A pool of hundreds of policies
  – Job size drawn uniformly at random
  – Deadline/value also drawn at random, proportionally to size
  – Spot price is a stochastic process; in each experiment, we change its characteristics
Simulations - synthetic data

- on-demand price is 0.25
- Experiment 1: Relatively low spot price (0.1+0.05x, where x is a Gaussian RV)
  - Outcome: Algorithm converges to using only spot instances with fixed bid=0.25.
Simulations - synthetic data

- **Experiment 2**: spot-price as before for first 10% of jobs, then becomes $0.2 + 0.05x$
  - Outcome: algorithm converges to using only on-demand instances
Simulations - synthetic data

• Experiment 3:
  – We let the spot price alternate between 0 for an hour and 0.3 for the next hour
  – Best policies are variable-bid policies with $\gamma = 0$ (price prediction based only on the last price)
Simulations – real data

• Setup:
  – “Translate” map-reduce jobs to our job model
  – Deadline taken as the job actual termination time from traces
  – Value added synthetically
Simulations – real data

• Spot-price as shown below
• Policy evolvement – first, alg uses both fixed and variable bid policies; however, when price becomes more stable, alg prefers fixed bid strategies
• Avg regret of our alg is 34 times better than the average regret
Related work

• Building statistical models for spot-prices [BBST13, JTB11]
• On-demand/spot assignment for bag of tasks [VOK13]
• Reserved instances [SDIE13]
Conclusion

• Online learning algorithm for choosing between on-demand and spot instances
  + No probabilistic assumptions
  + Incorporates a variety of policies
  + Incorporates job deadlines
  + Can be extended to many other resource allocation scenarios in cloud computing

• Future directions:
  – More scenarios
  – Including reserved instances