SPOTLYTICS: HOW TO USE CLOUD MARKET PLACES FOR DATA ANALYTICS?

TIM KRASKA, ELKHAN DADASHOV, CARSTEN BINNIG
**CLOUD IAAS**

**Idea:** Rent virtual machines from and run your software (e.g., DBMS, Spark, etc.)

**Typical Pricing Models**
- **On-demand:** fixed price per hour (e.g., 10 cent/hour)
- **Reserved:** basic fee based on contract over x years + lower hourly rate compared to on-demand
MARKET-BASED IAAS

IaaS providers overprovision their resources

Market-based IaaS: Overcapacity is sold under a dynamic pricing scheme

- **High Overcapacity** => Low Price
- **Low Overcapacity** => High Price (BUT also other parameters influence price)

**Main provider:** Amazon Spot Instances
AWS INSTANCES SPOT: USAGE MODEL

Bid Price $\geq$ Market Price: instance is granted

Bid Price $<$ Market Price: instance is not granted / revoked

Product: Linux/UNIX  
Instance type: c1.medium  
Date range: 1 month  
Availability zone: eu-west-1c

Market Price

Bid Price = 5 cent
AWS SPOT INSTANCES: PRICE MODEL

Prices are different per instance type + region + zone

Product: Linux/UNIX  Instance type: c1.medium  Date range: 1 month  Availability zone: eu-west-1c

Market Price  On-demand (no contract)

Reserved (3 years)
**AWS SPOT INSTANCES: BILLING**

**Billing** is based on an **interval** $\varepsilon$ (1h for Spot)

**Costs:** price at launch time*intervals (re-evaluated every interval)

**Discount:** for non-full intervals if instance is terminated by provider
CHALLENGES FOR ANALYTICS ON SPOT

Main goal should be to save monetary cost

Fault-tolerance of systems plays a key role

Other Peculiarities:

• all machines of the same type fail together
• weird almost binary (high price, low price) behavior
• price fluctuations for some types suddenly stopped
• abnormally high spikes
• etc.
PROBLEM STATEMENT

• **Given job J** (e.g., Map-Reduce program, a SQL query) and a **fault-tolerance strategy FT**

• Find the best deployment strategy to **minimize the overall monetary cost** of executing Q

---

**Deployment Strategy?**

- Type: 3 x m4.large
- Price: 5c / hour
COARSE-GRAINED RESTART

Scheme implemented in a Distributed DBMS

Recovery: Restart complete query
FINE-GRAINED RESTART + CHECKPOINTS

Scheme implemented in Hadoop

Recovery: Restart of individual operator instances
FINE-GRAINED RESTART + LINEAGE

Node 1

Node 2

Recovery: Restart of individual operator instances + lineage

Scheme implemented in Spark
CONTRIBUTIONS OF THIS PAPER?

Cost analysis for different fault-tolerance strategies

• Coarse-grained Query Restart
• Fine-grained Restart / Check pointing
• Fine-grained Restart / Lineage

Result 1. It is never beneficial to shut down an instance before the end of the billing interval $\varepsilon$. 
COARSE-GRAINED RESTART

Runtime costs of a job J (wo failure)

• Job is composed of multiple tasks
• Runtime of task on one instance: $R$
• Runtime of task on $n$ instances: $R/n$

On failure: Complete Restart

Result 2. Running a job in a single billing interval $\varepsilon$ is cheaper than running the job with fewer resources over several intervals
Result 2. Running a job in a single billing interval $\varepsilon$ is cheaper than running the job with fewer resources over several intervals.

- Assume that $q \cdot m$ is the number of machines to run the job in exactly one billing interval.
- Then $m$ the number of machines to run the job in $q$ intervals.
- Thus, cost for a successful run are equal.
- However, probability for failure increases with runtime $k$.

$$cn \left( e^{\frac{\lambda}{\varepsilon}} - 1 \right) \sum_{k=1}^{\varepsilon R} \left( e^{-\frac{\lambda(k-1)}{\varepsilon}} \left[ \frac{k}{\varepsilon} (1 - \gamma) + e^{-\frac{\lambda(k-1)}{\varepsilon}} \frac{k}{\varepsilon} \right] \right)$$
COARSE-GRAINED RESTART

Runtime costs of a Job J (wo failure)

• Job is composed of multiple tasks

• Runtime of task on one instance: \( R = \frac{R_{CPU}}{I_{CPU}} \) 
  \( (R_{CPU}: \text{Total Cycles, } I_{CPU}: \text{Cycles of instance in one } \varepsilon) \)

• Runtime of task on \( n \) instances: \( \frac{R}{n} \)

On failure: Complete Restart

**Result 2.** Running a job in a single billing interval \( \varepsilon \) is cheaper than running the job with fewer resources over several intervals

**Result 3.** Using more machines to finish early can be beneficial (depending on the failure rate \( \lambda \)).
EXP: VARYING # OF MACHINE

Low Failure Rate (\( \lambda = 0.75 \rightarrow \) every 800 minutes)

**Setup:** us-east-1c-m1.large–Linux instance type with on-demand price of $0.175 and a bid price of $0.0263 (15% of on-demand price)
EXP: VARYING # OF MACHINE

High Failure Rate (\(\lambda = 1.8\) -> every 33 minutes)

Setup: us-east-1c-m1.large–Linux instance type with on-demand price of $0.175 and a bid price of $0.0263 (15% of on-demand price)
Result 4. The expected cost of using $n$ or $2 \cdot n$ machines for a job is the “same” with check-pointing.

Intuition:

- Checkpointing allows to resume work “w/o loosing” invested work.
- Doubling machines reduces runtime by half but increases cost per billing interval by two.
**Result 4.** The expected cost of using \( n \) or \( 2 \cdot n \) machines for a job is the “same” with check-pointing.

**Intuition:**
- Checkpointing allows to resume work “w/o loosing” invested work.
- Doubling machines reduces runtime by half but increases cost per billing interval by two.

---

**Result 5.** Using a single instance to finish a job in a single check-pointing interval is the cheapest and most risk-averse option.

**Intuition:**
- High variance for one interval (i.e., pay nothing or all).
- Less variance for more intervals.
EXP: ONE VS. MANY MACHINES

Medium of the prices from 4 years as the bid-price

<table>
<thead>
<tr>
<th></th>
<th>1 instance for 100 hours</th>
<th>100 instance for 1 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{\mu}$</td>
<td>$\bar{\sigma}$</td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Setup: three machine types, m2.2xlarge, m2.4xlarge, and m2.xlarge all from the us-east-1a data center
Result 6. Same as Coarse-grained Query Restart on Spot Instances if we do not mix instance types
CONCLUSIONS

Market-based IaaS for Data Analytics

Main Contributions: Cost Analysis for different FT schemes

• Query Restart: Get more machines to pay less
• Fine-grained / Checkpointed (Hadoop): One machine saves most
• Fine-grained / Lineage (Spark): Same as query restart

Future work:

• Mixing instance types, bid prices for deployment
• Minimize runtime for given budget