Autonomic Provisioning and Application Mapping on Spot Cloud Resources

Daniel J. Dubois, Giuliano Casale

2015 IEEE International Conference on Cloud and Autonomic Computing (ICCAC), Cambridge, Massachusetts, September 21-25, 2015

Summarized by
Eniye Tebekaemi
Outline

• Spot Instance Pricing Models
• Problem Description
• System Model
• Proposed Solution
• Experimental Results
• Discussions
Spot Instance Pricing Model

• Spot instances enable you to bid on unused EC2 instances, which can lower your Amazon EC2 costs significantly.

• The hourly price for a Spot instance (of each instance type in each Availability Zone) is set by Amazon EC2, and fluctuates depending on the supply of and demand for Spot instances.

• Your Spot instance runs whenever your bid exceeds the current market price.

• Spot instances are a cost-effective choice if you can be flexible about when your applications run and if your applications can be interrupted.

Problem Description

• To take maximum advantage from spot instances users needs to make the following decisions:
  • What type of virtual resources should be rented for a given application?
  • How to efficiently map the components of an application (e.g., web server VMs, a database VMs) to the rented resources?
  • What is the lowest bid price that still allows to fulfill quality of service requirements?
• Users maybe SaaS providers or organizations running their applications and web services on the cloud.
Major Contributions

• Polynomial-time heuristic to jointly solve the bidding and allocation problem, which are in general NP-hard;

• Bidding in extended queueing network models that include a model of the operational environment, which captures the stochastic nature of the operational environment, in which VMs can be suddenly lost as a result of spot price fluctuations.

• The use of advanced fluid analysis techniques to accurately approximate response time percentiles, which are commonly used to constraint performance in service-level agreements.
System Model – Application

The Proposed system model consist of 2 parts: application and resources.

Application:

- Closed queuing network QN of M software servers.
- A delay node representing user think time
- K classes of requests
- Set of constraints on the response time defined as the SLO

Parameters

- $p_{m1,m2,k}$: Probability for a request of class $k$ to visit node $m2$ after completing service at node $m1$.
- $\mu_{m,k}$: Class service rate. Number of class-$k$ requests completed at software server $m$ in a time unit.
- $\sigma_k$: Delay node service rate. Number of class-$k$ requests completed at the delay node in a time unit.
- $N_k$: Total number of users of class $k$ in the system. Each user represents a request. This parameter specifies the system workload.
- $maxMRT_k$: Maximum mean response time for class-$k$ requests.
- $maxRTP_{k,u}$: Maximum response time for the class-$k$ requests in the $u$-th percentile.
System Model – Application

The Proposed system model consist of 2 parts: *application* and *resources*.

**Resources:**

- R + 1 available resource types. Type 0 is a special virtual type used to represent unallocated resources that have zero price and zero rate.
- Each resource is characterized by a certain rate (processing speed) and a certain number of processors.

**Parameters**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y ∈ ℤ⁺</td>
<td>Total number of resources that can be rented.</td>
</tr>
<tr>
<td>T</td>
<td>Rental time period.</td>
</tr>
<tr>
<td>A</td>
<td>Minimum percentage of time in which resources are expected to be available.</td>
</tr>
<tr>
<td>λ(r)</td>
<td>Nominal service rate of resources of type r. The value of λ(r) is calculated as the sum of the nominal service rate of each processor of the resource, and is a measure of the total computational capacity (e.g., it may be proportional to Amazon’s ECU [1]).</td>
</tr>
<tr>
<td>q(r)</td>
<td>Number of processors (CPUs) of resources of type r.</td>
</tr>
<tr>
<td>o(r)</td>
<td>Minimum bid price for renting a single resource of type r for a fixed amount of time T, such that the availability of the resource is at least A. This value can be obtained from historical traces of resource type r and calculated as the minimum bid price that results in an average outbid percentage of at least A for such price.</td>
</tr>
<tr>
<td>c(r)</td>
<td>Expected price for renting a single resource of type r for a fixed amount of time T when bidding o(r).</td>
</tr>
</tbody>
</table>
System Model — Decision Variable and Problem Statement

Our goal is to:

- Determine the rental and allocation policies, which consist in the amount of computational resources to be rented from a cloud provider, the mapping of the various application components to these resources, and finally the bid price for each resource.

- Decide a resource assignment vector \( t \) and an allocation matrix \( (d_{m,y}) \) that minimize the sum of the prices of all rented resources.

A formalization of the optimization problem is the following:

\[
\begin{align*}
\text{min} & \quad \sum_{y=1,\ldots,Y} \hat{c}_y \\
\text{s.t.} & \quad \sum_{m=1,\ldots,M} d_{m,y} \leq \hat{\lambda}_y, \forall y \\
& \quad MRT_k(D) \leq \max MRT_k, \forall k \\
& \quad RTP_{u,k}(D) \leq \max RTP_{u,k}, \forall u, \forall k
\end{align*}
\]
Proposed Solution

1. Choose the minimum computational requirements for each application component.

2. Choose the resources to rent – calculate the bidding price that guarantees a minimum availability level for the resources.

3. Choose the allocation of the application components to the resources

4. Analyze the overall system and possible scaling-up of bottlenecks.

State Diagram showing all the steps in the decision-making approach
Proposed Solution — Finding the optimal rate for each software server

1. **The minimum computational requirements in terms of resource rates that are needed by each software server to satisfy the SLO.**

2. **Assume that each software server is deployed on a dedicated hypothetical resource.**

3. **Each resource provides minimum rate to process request such that the SLO is satisfied.**

4. **The goal of this optimization problem is to decide the minimal rates** $\hat{\mu}_m$ **such:**

$$\min \sum_{m=1,\ldots,M} \hat{\mu}_m$$

s.t.  
$$MRT_k(\hat{\mu}) \leq \max MRT_k, \forall k$$

$$RTP_{u,k}(\hat{\mu}) \leq \max RTP_{u,k}, \forall u, \forall k$$

```python
1: function FINDOPTIMALRATES($\hat{\mu}_{init}$, $S$)
2:     $\hat{\mu}, \hat{\mu}_{min} \leftarrow 0$
3:     $\hat{\mu}_{max} \leftarrow \hat{\mu}_{init}$
4:     $r = 1, \ldots, S.M$     \Comment{Set of undecided rates}
5:     while $r \neq \emptyset$ do
6:         $\hat{\mu}(r) \leftarrow (\hat{\mu}_{min}(r) + \hat{\mu}_{max}(r))/2$
7:         if $SLO$satisfied($\hat{\mu}, S$) then
8:             $\hat{\mu}_{max} \leftarrow \hat{\mu}$
9:         else
10:            $vc = findViolatedClasses(\hat{\mu}, S)$
11:            $bn \leftarrow findBottleneckForClasses(r, \hat{\mu}, S, vc)$
12:            $\hat{\mu}_{min}(bn) \leftarrow \hat{\mu}(bn)$
13:        end if
14:        if $\max(\hat{\mu}_{max}(r) - \hat{\mu}_{min}(r)) < \epsilon$ then
15:            $vc \leftarrow findClassClosestToSLO(\hat{\mu}, S)$
16:            $bn \leftarrow findBottleneckForClasses(r, \hat{\mu}, S, vc)$
17:            $r \leftarrow r - bn$
18:        end if
19:     end while
20:     $\hat{\mu} \leftarrow max Rates$
21: return $\hat{\mu}$
22: end function
```
Proposed Solution – Finding the real resources to rent

- Decide which real resources to rent to provide the required computational needs at minimal expense.

- We consider for each real resource $y$ a mean price equal to $\hat{\mathbf{c}}_y$, that can be obtained from historical traces given a certain level of desired availability.

- The goal is to minimize the sum of these costs while ensuring that the rates of all rented resources are large enough to allocate the rates found as the solution of the previous problem.

$$\min \sum_{y=1,\ldots,Y} \hat{\mathbf{c}}_y$$

s.t. $$\sum_{y\in1,\ldots,Y} \hat{\lambda}_y \geq \sum_{m\in1,\ldots,M} \hat{\mu}_m$$
Proposed Solution — Finding the allocation of the rate for each software server to the real resources

• Find a good allocation of each software server to the rented resources based on their rates.

• Can allocate multiple software server to a single resource, or replicate a single software server over multiple resources.

• The goal is to minimize the overhead due to load balancing by minimizing the number of associations \((a_{m,y})\) between software servers such that

\[
\begin{align*}
\min & \quad \sum_{m=1}^{M} \sum_{y=1}^{Y} a_{m,y} \\
\text{s.t.} & \quad a_{m,y} = \begin{cases} 1 & \text{if } d_{m,y} \neq 0, \forall m, \forall y \\ 0 & \text{if } d_{m,y} = 0, \forall m, \forall y \\ \sum_{y \in Y} d_{m,y} \geq \mu_m, \forall m \\ \sum_{m \in M} d_{m,y} \leq \lambda_y, \forall y 
\end{cases}
\end{align*}
\]

```c
1: function FIND_RATE_ALLOCATION(\(\hat{\mu}, \hat{\lambda}\))
2: \(d \leftarrow 0\)
3: while max \(\hat{\mu} > 0\) do
4: \(m_{\text{max}} \leftarrow \text{argmax}_m \hat{\mu}(m)\)
5: \(y_{\text{max}} \leftarrow \text{argmax}_y \hat{\lambda}(y)\)
6: transfer \(\leftarrow \min(\hat{\mu}(m_{\text{max}}), \hat{\lambda}(m_{\text{max}}))\)
7: \(d(m_{\text{max}}, y_{\text{max}}) \leftarrow d(m_{\text{max}}, y_{\text{max}}) + \text{transfer}\)
8: \(\hat{\mu}(m_{\text{max}}) \leftarrow \hat{\mu}(m_{\text{max}}) - \text{transfer}\)
9: \(\hat{\lambda}(y_{\text{max}}) \leftarrow \hat{\lambda}(y_{\text{max}}) - \text{transfer}\)
10: end while
11: return \(d\)
12: end function
```
Proposed Solution — System analysis and scaling-up of the bottleneck server

- Check if the SLO constraints still hold when considering the system allocated using the resource assignment vector $t$ and the allocation matrix $D$ found in the previous steps.

- If SLO constraints still holds then apply solution.

- If the SLO constraints do not hold anymore then find the bottleneck server $m_*$.

- Scale up the bottleneck software server rates by a factor $\alpha$.

- Go back to step 2.

```c
1: function FINDBOTTLENECKM(\mu, S, \alpha)
2: bestSLOcompliance \leftarrow -\infty
3: m_* = \emptyset
4: for m \in 1, \ldots, S.M do
5: \hat{\mu}_{imp} \leftarrow \hat{\mu}
6: \hat{\mu}_{imp}(m) \leftarrow \hat{\mu}_{imp}(m) \times \alpha
7: t \leftarrow findResourcesToRent(\hat{\mu}_{imp}, S)
8: \hat{\lambda} = S.\lambda(t)
9: d = findRateAllocation(\hat{\mu}_{imp}, \hat{\lambda})
10: SLOcomp \leftarrow calcSLOcompliance(\hat{\mu}_{imp}, d, S)
11: if SLOcomp > bestSLOcomp then
12: bestSLOcompliance \leftarrow SLOcomp
13: m_* = \{m\}
14: else if SLOcomp = bestSLOcomp then
15: m_* = m_* \cup \{m\}
16: end if
17: end for
18: return m_*
19: end function
```
Evaluation – Settings

Hardware and Software
• 2.5 GHz Intel Core i7 quad-core processor with 16 GB of RAM running OS X 10.10.3 and MATLAB R2015a.
• LINE software to predict the response times of our queueing network
  • LINE can compute percentiles of response times, which are important for SLA assessment.

Application Model
• Application model based on previous measurements of an industrial enterprise resource planning (ERP) application, SAP ERP.
• Represented as a queueing network with exponentially distributed service times,
  • \( M = 2 \) software servers
  • \( K = 3 \) classes of requests

Resource Model
• Amazon EC2 historical spot price traces for each type of resources
• Computes resource rates as elastic computing unit (ECU)
  • 1 ECU = 65.1 requests/sec
**Evaluation – Parameters**

<table>
<thead>
<tr>
<th>Server/class</th>
<th>Service demand [ms]</th>
<th>Service rate $\mu_{m,k}$ [req/ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS dialog step</td>
<td>119.82</td>
<td>0.008346</td>
</tr>
<tr>
<td>AS update1</td>
<td>47.92</td>
<td>0.02087</td>
</tr>
<tr>
<td>AS update2</td>
<td>32.98</td>
<td>0.03032</td>
</tr>
<tr>
<td>DB dialog step</td>
<td>4.541</td>
<td>0.2202</td>
</tr>
<tr>
<td>DB update1</td>
<td>1.205</td>
<td>0.8299</td>
</tr>
<tr>
<td>DB update2</td>
<td>0.3043</td>
<td>3.286</td>
</tr>
</tbody>
</table>

**SAP ERP Parameters**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Rate ECU</th>
<th>CPUs</th>
<th>$r$</th>
<th>$\lambda_r$</th>
<th>$q_r$</th>
<th>Max bid to have $A = 90%$</th>
<th>Actual prices when $A = 90%$</th>
<th>Max bid to have $A = 95%$</th>
<th>Actual prices when $A = 95%$</th>
<th>Max bid to have $A = 99.9%$</th>
<th>Actual prices when $A = 99.9%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>1</td>
<td>1</td>
<td></td>
<td>0.067</td>
<td></td>
<td>0.0653</td>
<td>0.068</td>
<td>0.0659</td>
<td>0.07</td>
<td>0.067</td>
<td></td>
</tr>
<tr>
<td>m1.large</td>
<td>4</td>
<td>2</td>
<td></td>
<td>0.266</td>
<td></td>
<td>0.260</td>
<td>0.271</td>
<td>0.2622</td>
<td>0.28</td>
<td>0.2672</td>
<td></td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>8</td>
<td>4</td>
<td></td>
<td>0.534</td>
<td></td>
<td>0.5204</td>
<td>0.538</td>
<td>0.5222</td>
<td>0.559</td>
<td>0.5333</td>
<td></td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>6.5</td>
<td>2</td>
<td></td>
<td>0.325</td>
<td></td>
<td>0.3139</td>
<td>0.329</td>
<td>0.3164</td>
<td>0.336</td>
<td>0.3203</td>
<td></td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>13</td>
<td>4</td>
<td></td>
<td>0.735</td>
<td></td>
<td>0.7148</td>
<td>0.737</td>
<td>0.7157</td>
<td>0.769</td>
<td>0.7337</td>
<td></td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>26</td>
<td>8</td>
<td></td>
<td>1.468</td>
<td></td>
<td>1.4321</td>
<td>1.47</td>
<td>1.4342</td>
<td>1.54</td>
<td>1.468</td>
<td></td>
</tr>
</tbody>
</table>

**Spot Prices Of Amazon EC2**

---

*Monday, October 5, 2015*

CS 788: Autonomic Computing
Results

Experiment results when varying the number of users.
Experiment results when varying SLO.
The SLO is a maximum limit on the mean value of the response time calculated for a rental time period $T$. 
Results

Experiment results when varying the availability
Conclusion

• This paper shows that their approach is able to outperform an exact algorithm that is based on the MATLAB fmincon interior-point solver.

• Approximate a very complex problem using simple greedy algorithms that are lightweight enough to be used at run-time to support pro-active and reactive system adaptation.

• Predict and make decisions also when we have a representation of random environmental parameters such as the possibility for spot resources to be lost.