A Theory of Auto-Scaling for Resource Reservation in Cloud Services

Konstantinos Psychas, Javad Ghaderi
Columbia University

Performance 2020: 38th International Symposium on Computer Performance, Modeling, Measurements and Evaluation
November 2-6 2020, Milan(Online)
Problem Setting

Clients of different priorities

Requests (no queue)

Scheduler

Infrastructure

- We will use terms job and server
- Both treated as abstract multidimensional resource vectors
Model

If and where to schedule

A set of $J$ job types $\mathcal{J}$

Requests

Type $j \in \mathcal{J}$ gives reward $u_j$ per time unit

A set of $L$ homogeneous servers

Normalized Workload (jobs per server)

$\rho := (\rho_1, \rho_2, \ldots, \rho_J)$

#jobs from each type that can simultaneously fit in the server

$k = (k_1, k_2, \ldots, k_J) \in \mathcal{K} \subset \mathbb{R}^J$
How to schedule when workload is known

\[
\max_{X, Y} \sum_j u_j Y_j
\]

s.t. \( Y_j \leq \hat{Y}^L_j, \forall j \in J \)

\[
\sum_{k \in K} X_k k_j \geq Y_j, \forall j \in J
\]

\[
\sum_{k \in K} X_k = L, \quad X_k \geq 0, \forall k \in K
\]

Example \( \hat{Y} = L \rho \)

- Even if relaxed still hard (exponential number of variables)
- Needs to be solved every time workload changes
- Unclear how to change assignment when servers are already full
- There can be consistent error in \( \rho \) causing loss
Solving Static Optimization: Ordering Configuration by total Reward

- Reward of configuration
  \[ U(k) = \sum_{j=1}^{J} u_j k_j \]

- \( U(k) \) induces an ordering for all \( k \in \mathcal{K} \)

- Define for \( \mathcal{K}_s \subseteq \mathcal{K} \)
  \[ \text{MaxReward}(\mathcal{K}_s) = \arg \max_{k \in \mathcal{K}_s} U(k) \]
Solving Static Optimization: Greedy Placement Algorithm (GPA)

\[ \mathcal{K}^{\mathcal{J}^*} : \text{subset of configurations with jobs in } \mathcal{J}^* \]

\[ \text{Iteration } i \]
\[ k[i] := \text{MaxReward}(\mathcal{K}^{\mathcal{J}^*}) \]

Stop when \( \mathcal{J}^* \) empties or when \( L \) servers are assigned

Return \( \hat{\mathcal{X}} = \{ \hat{x}_{k[i]}, i = 1, \ldots, I \} \)
How good is greedy?

Consider normalized rewards ($L \to \infty$)

- Optimal Reward $U^*[\rho]$
- Greedy Reward $U^{(g)}[\rho]$

Without extra assumptions

Monotone Greedy Property

- If $\rho_1 \geq \rho_2$ then $U^{(g)}[\rho_1] \geq U^{(g)}[\rho_2]$

\[
U^{(g)}[\rho] \geq \frac{1}{2} U^*[\rho]
\]

\[
U^{(g)}[\rho] \geq (1 - e^{-1}) U^*[\rho]
\]
# Online Algorithm: Server Groups

<table>
<thead>
<tr>
<th>Server Group</th>
<th>Accept</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Try to Fill</td>
<td>Try to Empty</td>
</tr>
<tr>
<td>Schedule in it</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Migration</td>
<td>Migrate from a <strong>Reject</strong> Group filled slot to an <strong>Accept</strong> Group slot that empties</td>
<td></td>
</tr>
</tbody>
</table>
Online Algorithm: (CRA)
Classification and Reassignment Algorithm

- Get solution of GPA
  - Input $\hat{Y}^L = Y^L +/ - \text{arrival/departure} + g(L)1_j$
  - $g(L) = \omega(\log L)$: reservation factor
- Match server assignment to GPA solution
  - Matches configurations in decreasing reward order
CRA Example [Iteration $i$]

Goal: $\hat{X}_{k[i]}^L \leq X_{k[i]}^L$

Reject group if algorithm terminates

$\{k: \hat{X}_{k}^L = 0\}$
Dynamic Reservation Algorithm (DRA)

On arrival
- Run CRA
- In which slot to deploy the job arrived
- Answer: Any empty slot in Accept Group if exists

On departure
- Run CRA
- Which job to migrate in the slot that emptied
- Answer: Any job in a slot of Reject Group if exists
Informal Main Result

- Fraction of servers in each configuration of DRA $\rightarrow$ Fraction of servers in each configuration of GPA for input $\rho L$ when $L \rightarrow \infty$
- Normalized Reward of DRA $\rightarrow$ Greedy Reward $U^{(g)}[\rho]$
Simulations (Testing GPA approximation)

Generated 50 random setups
- 6VM types one per pair
- Rewards: 8vCPU + GB
- Servers: 80vCPU, 640GB
- Normalized workload $\rho_j$ in $[0.2, 2.0]$

Computed $U^{(g)}[\rho]/U^*[\rho]$ for each setup
- Worst $\approx 0.86$ [worst in theory 0.5]
- Average $\approx 0.97$
- Optimal = 1.00 [23/50 setups]

<table>
<thead>
<tr>
<th>Memory: GB per CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory-Opt</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>8 or 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>vCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
</tr>
<tr>
<td>2 or 4 or 8</td>
</tr>
</tbody>
</table>
Simulations (Testing with Google Trace)

- 1 million tasks
- 3 priorities
- 8 different sizes
- $3 \times 8 = 24$ types
Thank You