Generalizing Resource Allocation for the Cloud

Anshul Rai, Ranjita Bhagwan, Saikat Guha
Microsoft Research India
Resource Allocation Scenario

- Capacity-based VM Allocation
- Security domains
- Availability domains

Resource Allocation problems keep changing and adapting
*-Allocation

• VM Allocation
• Storage Allocation
• VLAN Allocation
• IP Address Space Allocation
• Server Allocation
• Network Allocation
Current approach

• Resource Management Tools (VMware, Microsoft, etc)
  – Implement their own heuristics
  – Often, not exactly what the administrator needs

• Custom Heuristics
  – Write and test the heuristics code
  – Change the code, repeat testing every time allocation constraints change.
  – Sometimes, constraints start conflicting. Heuristics difficult in such scenarios.
Why not consolidate?

- All these problems are variants of bin-packing
- So why not build a generic resource allocation service
- Reduces the pain of designing, writing, testing and extending custom heuristics
Solver-based Allocation

• Constraint-based programing
  – Z3, Kodkod, eCLiPse

• Built our first version of allocation service
  – Used Z3 and eCLiPse
  – Tough to write constraints
  – Too slow in a number of cases
Wrasse (Resource Allocation Service)

- Tough to write constraints
- Front End: “Balls and Bins” abstraction
- Too slow
- Back End: GPU-based solution generation
Wrasse Abstraction

BALLS: Virtual Machines
1 2 3 4 5 6 7 8 9

BINS: Servers
1 2 3 4 5 6

RESOURCES
Wrasse Abstraction

BALLS: Virtual Machines

1 2 3 4 5 6 7 8 9

BINS: Servers

1 2 3 4 5 6

RESOURCES

Server 1 Server 2
CPU CPU
capacity capacity
Wrasse Abstraction

BALLS: Virtual Machines
1 2 3 4 5 6 7 8 9

BINS: Servers

RESOURCES

| Server 1 CPU capacity | Server 2 CPU capacity | Server 6 CPU capacity | Link 1 Bandwidth | Link 2 Bandwidth |
Resource Utilization Function

• If Ball X goes into Bin Y, which resources are used, and by how much?
  – Depends on the allocation so far
Resource Utilization Function

BALLS: Virtual Machines

BINS: Servers

RESOURCES

<table>
<thead>
<tr>
<th>Server 6 CPU capacity</th>
<th>Link 1 Bandwidth</th>
<th>Link 2 Bandwidth</th>
<th>Link 3 Bandwidth</th>
<th>Link 5 Bandwidth</th>
<th>Link 6 Bandwidth</th>
<th>Link 8 Bandwidth</th>
</tr>
</thead>
</table>
Abstraction

• Declare: balls, bins and resources with their capacities
• Write: Resource allocation function.
VM Placement Specification

1: BALLS: \{0 \rightarrow VM0; 1 \rightarrow VM1; 2 \rightarrow VM2; 3 \rightarrow VM3\}
2: BINS: \{0 \rightarrow S0; 1 \rightarrow S1\}
3: RESOURCES: \{0 \Rightarrow (S0CPU, 100); 1 \Rightarrow (S0MEM, 5); 2 \Rightarrow (S1CPU, 200); 3 \Rightarrow (S1MEM, 10)\}
5:
6: \textbf{procedure} UTILFn(BALL, BIN, ALLOC)
7: \textbf{UTILData}: \{0 \Rightarrow 100; 1 \Rightarrow 2; 2 \Rightarrow 50; 3 \Rightarrow 3; 4 \Rightarrow 40; 5 \Rightarrow 4; 6 \Rightarrow 40; 7 \Rightarrow 4\}
11: \textbf{UTIL} \leftarrow \{0, 0, 0, 0\}
12: UTIL[BIN \times 2] \leftarrow UTILData[BALL \times 2]
13: UTIL[BIN \times 2 + 1] \leftarrow UTILData[BALL \times 2 + 1]
14: \textbf{return UTIL}
15:
16: \textbf{FOES}: [{VM2, VM3}]
Friends, Foes and Pinning

• Friends
  – Always put them on the same bin

• Foes
  – Put at least one of the foes in a different bin

• Pin
  – Pin ball X on bin Y
  – Important for incremental changes
Soft constraints

• “Satisfy friend constraint with a probability of 90%”
• “Allow Server 1’s CPU capacity to go above limit by 10% with a probability of 5%”
Evolving the Allocation Spec

1: \textbf{Balls}: \{0 \Rightarrow \text{VM0}; 1 \Rightarrow \text{VM1}; 2 \Rightarrow \text{VM2}; 3 \Rightarrow \text{VM3}\}
2: \textbf{Bins}: \{0 \Rightarrow \text{S0}; 1 \Rightarrow \text{S1}\}
3: \textbf{Resources}: \{0 \Rightarrow (\text{S0Cpu, 100}); 1 \Rightarrow (\text{S0Mem, 5});
\quad 2 \Rightarrow (\text{S1Cpu, 200}); 3 \Rightarrow (\text{S1Mem, 10})\}
5:
6: \textbf{procedure} \text{UtilFn}(\text{Ball, Bin, Alloc})
7: \qquad \textbf{UtilData}: \{0 \Rightarrow 100; 1 \Rightarrow 2; \}
8: \quad 2 \Rightarrow 50; 3 \Rightarrow 3; \}
9: \quad 4 \Rightarrow 40; 5 \Rightarrow 4; \}
10: \quad 6 \Rightarrow 40; 7 \Rightarrow 4\}
11: \quad \text{Util} \leftarrow \{0, 0, 0, 0\}
12: \quad \text{Util}[\text{Bin} \times 2] \leftarrow \text{UtilData}[\text{Ball} \times 2]
13: \quad \text{Util}[\text{Bin} \times 2 + 1] \leftarrow \text{UtilData}[\text{Ball} \times 2 + 1]
14: \quad \textbf{return} \ \text{Util}
15: \quad \textbf{return} \ \text{Util}
16: \textbf{Foels}: \{\text{VM2, VM3}\}
Evolving the Allocation Spec
SecondNet: Network Virtualization

1: \textbf{RESOURCES:} \{\ldots, 4 \Rightarrow (\text{LINK0}, 150), 5 \Rightarrow (\text{LINK1}, 100)\}

2: \textbf{procedure} \textit{UtilFn}(\textit{BALL, BIN, ALLOC})

3: \textit{BW}: \{0 \Rightarrow \{0, 10, 0, 0\};
4: 1 \Rightarrow \{20, 0, 0, 0\};
5: 2 \Rightarrow \{0, 0, 0, 50\};
6: 3 \Rightarrow \{0, 0, 50, 0\}\}

7: \textit{PATH}: \{0 \Rightarrow \{1 \Rightarrow \{4, 5\}\};
8: 1 \Rightarrow \{0 \Rightarrow \{5, 4\}\}\}

9: \textbf{for all \textit{OBALL} in 0\ldots3 except \textit{BALL} do}
10: \hspace{1em} \textit{OBIN} \leftarrow \text{ALLOC}[\textit{OBALL}]
11: \hspace{1em} \textbf{if} \textit{OBIN} \neq \textit{NULL} \textbf{and} \textit{OBIN} \neq \textit{BIN} \textbf{then}
12: \hspace{2em} \textbf{for all \textit{LINK} in \textit{PathToLCA}\{\textit{BIN}\}[\textit{OBIN}] \textbf{do}
13: \hspace{3em} \text{\textit{UTIL}}[\text{\textit{LINK}}] \leftarrow \text{\textit{BW}}[\text{\textit{BALL}}][\text{\textit{OBALL}}] + \text{\textit{BW}}[\text{\textit{OBALL}}][\text{\textit{BALL}}]
14: \hspace{2em} \textbf{if} \textit{OBIN} \neq \textit{NULL} \textbf{then}
15: \hspace{3em} \textbf{for all \textit{LINK} in \textit{PathToRoot}\{\textit{BIN}\} \textbf{do}
16: \hspace{4em} \text{\textit{UTIL}}[\text{\textit{LINK}}] \leftarrow \text{\textit{BW}}[\text{\textit{BALL}}][\text{\textit{OBALL}}] + \text{\textit{BW}}[\text{\textit{OBALL}}][\text{\textit{BALL}}]
17: \hspace{3em} \textbf{if} \textit{OBIN} \neq \textit{NULL} \textbf{and} \textit{OBIN} \neq \textit{BIN} \textbf{then}
18: \hspace{4em} \textbf{for all \textit{LINK} in \textit{PathToRoot}\{\textit{BIN}\} \textbf{do}
19: \hspace{5em} \text{\textit{UTIL}}[\text{\textit{LINK}}] \leftarrow \text{\textit{BW}}[\text{\textit{BALL}}][\text{\textit{OBALL}}] + \text{\textit{BW}}[\text{\textit{OBALL}}][\text{\textit{BALL}}]

\ldots
A Discussion on this Design

• Balls of only one type, bins of only one type
• No notion of a network
• As a result, resource utilization function can get complicated
• But simplicity important for solver implementation.

Can we model different kinds of balls?
Can we model different kinds of bins?
Can we model resource utilizations other than additive?
Back End: GPU-based solver

- Pick a ball at random
- Put it in the first bin
- Satisfy all Friend-Foe constraints
- Use resource utilization function to ensure no resource capacities are exceeded
- Pick another ball … until all balls have been tried for this bin.
Explore vs Exploit

Solution space potentially explored after

- bin0
- bin1
- bin2
GPU Implementation

• Version 1: Each thread finds a potential solution (16 solutions simultaneously checked)
  – Memory issues
  – Scale issues

• Version 2: Each thread-group finds a potential solution (4 solutions simultaneously checked)
VM Placement

Input

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PgRank</td>
<td>474</td>
<td>0.16</td>
<td>2.94</td>
<td>7.67</td>
<td>1.95</td>
<td>2.03</td>
</tr>
<tr>
<td>ClkBt</td>
<td>885</td>
<td>0.14</td>
<td>1.07</td>
<td>19.69</td>
<td>0.78</td>
<td>1.22</td>
</tr>
<tr>
<td>ImgProc</td>
<td>2942</td>
<td>0.37</td>
<td>0.35</td>
<td>1.41</td>
<td>0.92</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Solution quality (comparing to SCVMM heuristics)

<table>
<thead>
<tr>
<th>Application</th>
<th>FFDProd</th>
<th>DotProd</th>
<th>NBG</th>
<th>Z3</th>
<th>Wrasse</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgRank</td>
<td>90</td>
<td>100</td>
<td>97</td>
<td>90</td>
<td>89</td>
</tr>
<tr>
<td>ClkBt</td>
<td>420</td>
<td>420</td>
<td>420</td>
<td>424</td>
<td>420</td>
</tr>
<tr>
<td>ImgProc</td>
<td>1406</td>
<td>1403</td>
<td>1406</td>
<td>1417</td>
<td>1403</td>
</tr>
</tbody>
</table>

Solution time (ms)

<table>
<thead>
<tr>
<th>Application</th>
<th>FFDProd</th>
<th>DotProd</th>
<th>NBG</th>
<th>Z3</th>
<th>Wrasse</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgRank</td>
<td>7</td>
<td>16.2</td>
<td>19.2</td>
<td>30864</td>
<td>51</td>
</tr>
<tr>
<td>ClkBt</td>
<td>15.6</td>
<td>69.2</td>
<td>82.6</td>
<td>146149</td>
<td>7645</td>
</tr>
<tr>
<td>ImgProc</td>
<td>92.6</td>
<td>744.2</td>
<td>923.6</td>
<td>139876</td>
<td>370</td>
</tr>
</tbody>
</table>
Network Virtualization

SecondNet (CoNext 2010)

1024 servers, 2-level fat-tree.
Average Virtual Data Center (VDC) size: 94.
Keep assigning VDCs until assignment fails
Performance: GPU vs CPU

• Used AMD HD6990 and the nVidia Tesla

• Tesla implementation worked about 8.5 times faster than 3 GHz Intel Core 2 Duo processor
Related Work

• Rhizoma: Used eCLiPse for configuration management
  – Runs into performance issues with large-sized problems.

• Cologne: Distributed platform for configuration management
  – Uses constraint solvers as well in the back-end.

• Various heuristic-based solutions for configuration management
  – Wrasse can encode all that we have encountered.
Summary

• Presented a generic resource allocation service for the cloud

• Good performance, both in terms of time to run and solution quality

• We have built a web service around Wrasse so it can be easily used
Questions?