Colocation Explained

Heartbeat 2.1.2-4 Onwards

abeekhof@suse.de
Terminology

- Collocate(B, A)
- <rsc_colocation from=B to=A/>
- Decide where to put A, then put B there too
- Include B’s preferences when deciding where to put A
- If A cannot run anywhere, B can’t run either
- If B cannot run anywhere, A will be unaffected
Adding Scores

- number > INFINITY = INFINITY
- number < -INFINITY = - INFINITY
- number + INFINITY = INFINITY
- number - INFINITY = - INFINITY
- INFINITY - INFINITY = - INFINITY

- INFINITY ::= 1,000,000
Simple Example

**Setup**

- `resource(A, priority=5)`
- `resource(B, priority=50)`
- `location(A, node1, 100)`
- `location(A, node2, 10)`
- `location(B, node2, 1000)`
- `collocate(B, A)`
Simple Example

What Happens

• Start at highest priority resource (B)
• Defer and process A instead (collocation rule)
• Incorporate B’s preferences
  • A.node1.score += B.node1.score (100)
  • A.node2.score += B.node2.score (1010)
• Choose a node (node2)
Simple Example
Actually I Lied

• Incorporate B’s preferences
  • A.node[x].score += \textit{factor} \times B.node[x].score
• What is \textit{factor}?
  • \textit{factor} ::= \textit{constraint.score} / \text{INFINITY}
• For most people it will be 1 or -1
• So really its: colocate(B, A, \textit{score})
Choosing a Node for B

Simple Example

- Process collocation constraint
  - Matching node: node.score = INFINITY
  - Everything else: node.score = -INFINITY
- Scores do not include A’s preferences
- Final scores for B
  - node1 = -INFINITY
  - node2 = INFINITY
Choosing a Node for **B**

**Suggested Colocation**

- When the collocation score != INFINITY
  - Matching node: node.score += collocation.score
  - Everything else: unchanged
- Scores do **not** include **A**’s preferences
- Final scores for **B** (collocation.score = 500)
  - node1 = 0
  - node2 = 1500
Chained Example

Setup

- resource(A, p=5)
- resource(B, p=500)
- resource(C, p=50)
- location(A, node1, 100)
- location(A, node2, 10)
- collocate(B, A)
- collocate(C, B)
- location(B, node2, 1000)
- location(C, node1, 10000)
Chained Example

What Happens

- Start at highest priority resource (B)
- Defer and process A instead (collocation rule)
- Incorporate B’s preferences
  - A.node[x].score += B.node[x].score
- So far nothing is different
Chained Example

What Happens (Continued)

- Incorporate C’s preferences too!
  - A.node[x].score += C.node[x].score

- Final scores (when choosing a node for A)
  - node1 = 10100
  - node2 = 1010
Chained Example

Final Scores: B and C

- Resource B
  - node1 = INFINITY
  - node2 = -INFINITY

- Resource C
  - node1 = INFINITY
  - node2 = -INFINITY
Multiple Dependencies

• Include scores from B, C and D when choosing a node for A

• Order is defined by priority of dependent resources (or name if priority is equal)

• In this example:
  • B.priority > C.priority
  • C.priority > D.priority
Dependancy Tree

Order in Which Preferences are Applied (A-H)
More Complex

C is a Group

Diagram:
- F → E → C3 → C2 → C1 → A
- D → C3
- B → C1
- G → C3
- H → C1
Getting Smart
When not Everything can Run

• If applying a resource’s preference, means that all nodes would be unavailable...
  • Undo the current resource’s preference
  • Skip any resources that need to be collocated with the current resource
  • Process the next peer
Un-runnable: B
Un-runnable: B
Un-runnable: E

Diagram:
- A
- B
- C
- D
- E
- F
- G
- H

Connections:
- A -> B
- A -> C
- C -> D
- C -> E
- E -> F
- E -> G
- B -> H
- H -> C

Un-runnable: E

Diagram with nodes labeled F, D, B, C, A, G, H, and an 'X' indicating an un-runnable path.
Un-runnable: E
Un-runnable: C
Un-runnable: C
Un-runnable: C
Un-runnable: C

Diagram showing nodes A, B, D, G, and H, with some nodes marked with an 'X'.
Un-runnable: C
Un-runnable: C
## Un-runnable

### Worked Example

<table>
<thead>
<tr>
<th>Rsc</th>
<th>Node</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>node1</td>
<td>50</td>
</tr>
<tr>
<td>A</td>
<td>node2</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>node1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>node2</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>node1</td>
<td>-INFINITY</td>
</tr>
<tr>
<td>C</td>
<td>node2</td>
<td>-INFINITY</td>
</tr>
<tr>
<td>D</td>
<td>node1</td>
<td>100</td>
</tr>
</tbody>
</table>

Graph:
- A is connected to B, C, and D.
• Consider B
  • A.node1.score = 50 + 1
  • A.node2.score = 5 + 10
• Consider \textbf{C}
  • A.node1.score = 51 \textbackslash{}INFINITY
  • A.node2.score = 15 \textbackslash{}INFINITY

• Rollback Scores
  • A.node1.score = 51
  • A.node2.score = 15
• Consider C
  • A.node1.score = 51 -INFINITY
  • A.node2.score = 15 -INFINITY
• Rollback Scores
  • A.node1.score = 51
  • A.node2.score = 15
• Consider D
  • A.node1.score = 51 + 100
  • A.node2.score = 15 + 1000
• Final Scores
  • A.node1.score = 151
  • A.node2.score = 1015
• Choose node2
• Consider $D$
  • $A.node1.score = 51 + 100$
  • $A.node2.score = 15 + 1000$
• Final Scores
  • $A.node1.score = 151$
  • $A.node2.score = 1015$
• Choose $\text{node2}$
Colocation by Role
Master/Slave - Summary

• A resource that needs to run on the master can force the master to move (rather than not be allowed to run anywhere)

• A resource that can’t run anywhere and must run with the master does not prevent the promotion of a master
Colocation by Role

Who Gets Promoted

• Allocation occurs as-per previous slides

• Decision of which instances to promote is based on

  • Preference as set by RA with crm_master

  • **Location preferences of resources that wish to be colocated with the master instance(s)**
# Colocation by Role

## Master/Slave Example

<table>
<thead>
<tr>
<th>Child</th>
<th>Location</th>
<th>M/S Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms:0</td>
<td>node1</td>
<td>1,000</td>
</tr>
<tr>
<td>ms:1</td>
<td>node2</td>
<td>100</td>
</tr>
<tr>
<td>ms:2</td>
<td>node3</td>
<td>10</td>
</tr>
<tr>
<td>ms:3</td>
<td>node4</td>
<td>-INFINITY</td>
</tr>
</tbody>
</table>
Colocation by Role

Changes

• Under the old system, we would
  • sort the children by their m/s score
  • allocate masters in that order (ms:0, ms:1, ms:2)
• Now we include the colocation scores too
## Colocation by Role

### Master/Slave Example (continued)

<table>
<thead>
<tr>
<th>Dependent</th>
<th>Location</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsc1</td>
<td>node1</td>
<td>20</td>
</tr>
<tr>
<td>rsc2</td>
<td>node2</td>
<td>200</td>
</tr>
<tr>
<td>rsc3</td>
<td>node2</td>
<td>-INFINITY</td>
</tr>
<tr>
<td>rsc3</td>
<td>node3</td>
<td>2,000</td>
</tr>
<tr>
<td>rsc4</td>
<td>[everywhere]</td>
<td>-INFINITY</td>
</tr>
</tbody>
</table>
## Colocation by Role

### Master/Slave Example (continued)

<table>
<thead>
<tr>
<th>Child</th>
<th>Location</th>
<th>M/S Score</th>
<th>Final Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms:0</td>
<td>node1</td>
<td>1,000</td>
<td>1,020</td>
</tr>
<tr>
<td>ms:1</td>
<td>node2</td>
<td>100</td>
<td>-INFINITY</td>
</tr>
<tr>
<td>ms:2</td>
<td>node3</td>
<td>10</td>
<td>2,010</td>
</tr>
<tr>
<td>ms:3</td>
<td>node4</td>
<td>-INFINITY</td>
<td>-INFINITY</td>
</tr>
</tbody>
</table>
Colocation by Role

Master/Slave Example (continued)

• “Final” weight affects sorting order only

• Negative final score does not prevent the instance from being promoted

• Sort and allocate Masters in order (depending on the number of masters required):

  • ms:2, ms:0, ms:1

  • ms:3 can’t be promoted as it’s m/s score is less than zero
abeekhof@suse.de or linux-ha@lists.linux-ha.org