Constraint-Based Specifications for System Configuration

John A. Hewson

Feb 14, 2011
Overview

• Cloud and IaaS configuration
• State-of-the-art: Declarative languages
• Modelling an IaaS problem
• Solving with CSP
• Future work: Semantics, usability, advanced features
Configuration Errors Matter

Service disruption events by most likely cause at one of Google’s main services, over 6 weeks (2009)

*The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Hoelzle & Barroso, 2009.*
Cloud Computing = Platform as a Service (PaaS)

http://blog.softheme.com/cloud-services-convenient-for-small-business/

e.g. Amazon S3, Google AppEngine. Not off-site VMware or Xen.

Why? Because individual cloud machines are not meant to be reliable.
Infrastructure as a Service (IaaS)

• Saves on infrastructure costs (both CapEx and OpEx)
• VMware is used by 98% of Fortune 500 companies
  http://www.vmware.com/company/customers/
• Can even move running VMs in near-realtime
Configuration – solutions?

Rise of declarative tools for UNIX:

• LCFG (1993, Anderson, University of Edinburgh)
  – configures your DICE machine!
• Cfengine (1993, Burgess, Oslo University College)
• Bcfg2 (2004, Desai, Argonne National Laboratory)
• Puppet (2005, Kanies, Independent)
Puppet

• Used at major web companies: Twitter, match.com, Zynga
• Open Source (GPL)
• Configures UNIX-like systems, abstracting over differences
• Declarative language. For example, we write

```ruby
package {'apache':
    ensure => installed
}
```

instead of

```
sudo apt-get --yes install apache
```
What’s Missing?

• Constraints!
• The ability to *verify* that a configuration conforms to a model
• The ability to *infer* valid configurations from a model
  – Much more powerful
  – Now required for IaaS and cloud-scale systems, as the problems are too time-consuming for humans to solve.

• *Let’s look at an example*...
Some IaaS Problems in the Enterprise

• How can we assign VMs to physical machines?
  – With CPU, RAM, I/O requirements
  – With co-location requirements (e.g. distribute redundant VMs)
  – In Compliance (e.g. following credit card data rules)
  – Following Firewall rules (or changing them)

• How can we optimise:
  – The VM assignments above
  – Latency between pairs of machines
  – Power consumption
  – Licensing (e.g. per-CPU)
  – Robustness (e.g. redundancy)
  – Performance (e.g. database cache)
  – SLAs (e.g. minimise cost of violation)
Example: Problem

• Service–Machine Allocation
• 4 Services
• 3 Machines
• Each machine has a fixed:
  – Scalar amount of RAM
  – Scalar number of CPUs
  – Boolean set of capabilities (e.g. RAID5, Gigabit Ethernet)
• Each service has fixed requirements over these values

• Q: Which services run on which machines?
### Existing Machine Capabilities

<table>
<thead>
<tr>
<th>Machine</th>
<th>Capability</th>
<th>MachineCapabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monster</td>
<td>IsIISEnabled</td>
<td>0</td>
</tr>
<tr>
<td>Monster</td>
<td>IsSQLEnabled</td>
<td>1</td>
</tr>
<tr>
<td>Monster</td>
<td>HasDualProc</td>
<td>1</td>
</tr>
<tr>
<td>Monster</td>
<td>HasQuadProc</td>
<td>1</td>
</tr>
<tr>
<td>Monster</td>
<td>HasRAID5</td>
<td>1</td>
</tr>
<tr>
<td>Monster</td>
<td>HasGigEther</td>
<td>0</td>
</tr>
<tr>
<td>Chatter</td>
<td>IsIISEnabled</td>
<td>1</td>
</tr>
<tr>
<td>Chatter</td>
<td>IsSQLEnabled</td>
<td>0</td>
</tr>
<tr>
<td>Chatter</td>
<td>HasDualProc</td>
<td>0</td>
</tr>
<tr>
<td>Chatter</td>
<td>HasQuadProc</td>
<td>0</td>
</tr>
<tr>
<td>Chatter</td>
<td>HasRAID5</td>
<td>0</td>
</tr>
<tr>
<td>Chatter</td>
<td>HasGigEther</td>
<td>1</td>
</tr>
<tr>
<td>Typical</td>
<td>IsIISEnabled</td>
<td>0</td>
</tr>
<tr>
<td>Typical</td>
<td>IsSQLEnabled</td>
<td>0</td>
</tr>
<tr>
<td>Typical</td>
<td>HasDualProc</td>
<td>1</td>
</tr>
<tr>
<td>Typical</td>
<td>HasQuadProc</td>
<td>0</td>
</tr>
<tr>
<td>Typical</td>
<td>HasRAID5</td>
<td>1</td>
</tr>
<tr>
<td>Typical</td>
<td>HasGigEther</td>
<td>0</td>
</tr>
</tbody>
</table>

### Service Minimum Requirement (Capabilities)

<table>
<thead>
<tr>
<th>Service</th>
<th>Capability</th>
<th>ServiceCapabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omniscient</td>
<td>IsIISEnabled</td>
<td>0</td>
</tr>
<tr>
<td>Omniscient</td>
<td>IsSQLEnabled</td>
<td>1</td>
</tr>
<tr>
<td>Omniscient</td>
<td>HasDualProc</td>
<td>0</td>
</tr>
<tr>
<td>Omniscient</td>
<td>HasQuadProc</td>
<td>0</td>
</tr>
<tr>
<td>Omniscient</td>
<td>HasRAID5</td>
<td>1</td>
</tr>
<tr>
<td>Omniscient</td>
<td>HasGigEther</td>
<td>0</td>
</tr>
<tr>
<td>FrontEnd</td>
<td>IsIISEnabled</td>
<td>1</td>
</tr>
<tr>
<td>FrontEnd</td>
<td>IsSQLEnabled</td>
<td>0</td>
</tr>
<tr>
<td>FrontEnd</td>
<td>HasDualProc</td>
<td>0</td>
</tr>
<tr>
<td>FrontEnd</td>
<td>HasQuadProc</td>
<td>0</td>
</tr>
<tr>
<td>FrontEnd</td>
<td>HasRAID5</td>
<td>0</td>
</tr>
<tr>
<td>FrontEnd</td>
<td>HasGigEther</td>
<td>1</td>
</tr>
<tr>
<td>Industrious</td>
<td>IsIISEnabled</td>
<td>0</td>
</tr>
<tr>
<td>Industrious</td>
<td>IsSQLEnabled</td>
<td>0</td>
</tr>
<tr>
<td>Industrious</td>
<td>HasDualProc</td>
<td>1</td>
</tr>
<tr>
<td>Industrious</td>
<td>HasQuadProc</td>
<td>0</td>
</tr>
<tr>
<td>Industrious</td>
<td>HasRAID5</td>
<td>0</td>
</tr>
<tr>
<td>Industrious</td>
<td>HasGigEther</td>
<td>0</td>
</tr>
<tr>
<td>Schizoid</td>
<td>IsIISEnabled</td>
<td>0</td>
</tr>
<tr>
<td>Schizoid</td>
<td>IsSQLEnabled</td>
<td>0</td>
</tr>
<tr>
<td>Schizoid</td>
<td>HasDualProc</td>
<td>1</td>
</tr>
<tr>
<td>Schizoid</td>
<td>HasQuadProc</td>
<td>0</td>
</tr>
<tr>
<td>Schizoid</td>
<td>HasRAID5</td>
<td>0</td>
</tr>
<tr>
<td>Schizoid</td>
<td>HasGigEther</td>
<td>0</td>
</tr>
</tbody>
</table>

### Existing Machine Metric

<table>
<thead>
<tr>
<th>Machine</th>
<th>Metric</th>
<th>MachineMetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monster</td>
<td>Memory</td>
<td>16384</td>
</tr>
<tr>
<td>Monster</td>
<td>CPU</td>
<td>12</td>
</tr>
<tr>
<td>Chatter</td>
<td>Memory</td>
<td>1024</td>
</tr>
<tr>
<td>Chatter</td>
<td>CPU</td>
<td>2</td>
</tr>
<tr>
<td>Typical</td>
<td>Memory</td>
<td>2048</td>
</tr>
<tr>
<td>Typical</td>
<td>CPU</td>
<td>3</td>
</tr>
</tbody>
</table>

### Service Minimum Requirement (Metric)

<table>
<thead>
<tr>
<th>Service</th>
<th>Metric</th>
<th>ServiceMetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omniscient</td>
<td>Memory</td>
<td>4096</td>
</tr>
<tr>
<td>Omniscient</td>
<td>CPU</td>
<td>6</td>
</tr>
<tr>
<td>FrontEnd</td>
<td>Memory</td>
<td>512</td>
</tr>
<tr>
<td>FrontEnd</td>
<td>CPU</td>
<td>1</td>
</tr>
<tr>
<td>Industrious</td>
<td>Memory</td>
<td>512</td>
</tr>
<tr>
<td>Industrious</td>
<td>CPU</td>
<td>1</td>
</tr>
<tr>
<td>Schizoid</td>
<td>Memory</td>
<td>1024</td>
</tr>
<tr>
<td>Schizoid</td>
<td>CPU</td>
<td>2</td>
</tr>
</tbody>
</table>
Example: Specification (Classes)

- System
  - FrontEnd
  - Omniscient
  - Industrious
  - Schizoid
  - Typical
  - Monster

- Service
  - required_cpu : int
  - required_memory : int

- Machine
  - cpu : int
  - memory : int

- Chatter

- Chatter

- runs_on
Example: Specification

```plaintext
component Machine { 
    var cpu as int;
    var memory as int;
}

component Service {
    var required_cpu as int;
    var required_memory as int;
    var runs_on as ref Machine;
}

...
Example: Specification

```java
component FrontEnd extends Service {
    where required_cpu == 1;
    where required_memory == 512;
    where required_capabilities == {IsIISEnabled, HasGigEther};
}

component Monster extends Machine {
    where required_cpu == 1;
    where required_memory == 512;
    where required_capabilities == {IsIISEnabled, HasDualProc, HasQuadProc, HasRAID5};
}
```

...
Example: Specification

```
root component System {
    var typical as Typical;
    var monster as Monster;
    var chatter as Chatter;
    
    var front_end as FrontEnd;
    var omniscient as Omniscient;
    var industrious as Industrious;
    var schizoid as Schizoid;
    
    ...
}
```
Example: Specification

```csharp
var machines as (ref Machine)[3];
var services as (ref Service)[4];

foreach (m in machines, s in services where s.runs_on == m) {
    sum(s.required_cpu) <= m.cpu &&
    sum(s.required_memory) <= m.memory &&
    s.required_capabilities in m.capabilities;
}
```
Example: Specification (Instances)
Constraint-Satisfaction Problem (CSP)

- Closely related to SAT and SMT solvers.
- Problem is described as a sets of variables, domains, and constraints.
- Everything is finite – complete, decidable. *Very desirable properties.*
- Modern solvers also support optimisation, local search, and soft constraints.
- $N$-queens problem: or Sudoku:

![Chessboard](The Code Project)

![Sudoku](http://radialmind.blogspot.com)
Auto-Generated CSP Code (MiniZinc)

/** variables */
var int : root_typical_cpu;
var int : root_typical_memory;
var int : root_monster_cpu;
var int : root_monster_memory;
var int : root_chatter_cpu;
var int : root_chatter_memory;
var int : root_front_end_required_cpu;
var int : root_front_end_required_memory;
var int : root_industrious_cpu;
var int : root_industrious_memory;
var int : root_omniscient_cpu;
var int : root_omniscient_memory;
var int : root_schizoid_cpu;
var int : root_schizoid_memory;

/** constraints */
/* System */
constraint (((bool2int((root_front_end_runs_on = 2)) * root_front_end_required_cpu) + (bool2int((root_omniscient_runs_on = 2)) * root_omniscient_required_cpu) + (bool2int((root_industrious_runs_on = 2)) * root_industrious_required_cpu)) <= root_monster_cpu);

constraint (((bool2int((root_front_end_runs_on = 2)) * root_front_end_required_memory) + (bool2int((root_omniscient_runs_on = 2)) * root_omniscient_required_memory) + (bool2int((root_industrious_runs_on = 2)) * root_industrious_required_memory)) <= root_monster_memory);

constraint (((bool2int((root_front_end_runs_on = 1)) * root_front_end_required_cpu) + (bool2int((root_omniscient_runs_on = 1)) * root_omniscient_required_cpu) + (bool2int((root_industrious_runs_on = 1)) * root_industrious_required_cpu)) <= root_chatter_cpu);

constraint (((bool2int((root_front_end_runs_on = 1)) * root_front_end_required_memory) + (bool2int((root_omniscient_runs_on = 1)) * root_omniscient_required_memory) + (bool2int((root_industrious_runs_on = 1)) * root_industrious_required_memory)) <= root_chatter_memory);

constraint ((root_typical_cpu = 3) / (root_typical_memory = 2848));
constraint ((root_monster_cpu = 12) / (root_monster_memory = 16384));
constraint ((root_chatter_cpu = 2) / (root_chatter_memory = 1824));
constraint ((root_front_end_required_cpu = 1) / (root_front_end_required_memory = 512));
constraint ((root_omniscient_cpu = 6) / (root_omniscient_required_memory = 496));
constraint ((root_industrious_required_cpu = 1) / (root_industrious_required_memory = 512));
constraint ((root_schizoid_required_cpu = 2) / (root_schizoid_required_memory = 1024));
solve satisfy;
CSP Solution

• Used the *Gecode* CSP Solver, which supports:
  – Backtracking search
  – Local search
  – Optimisation functions
  – Decision heuristics

• Takes < 400ms (hard to benchmark tiny problems)
• *Lets show the solution visually...*
Example: Problem (Instances)
Example: Solution (Instances)
On-Going & Future Work

• Formally defined semantics for the configuration language, including:
  – Refinement Types (*e.g.* x:int where x > 4)
  – Optimisation Functions
  – Soft Constraints (Preferences)

• Minimum-change goal (for Re-Configuration)

• Usability

• Generate *Puppet* code using templates
Summary

• Cloud and IaaS configuration
  – Cloud = PaaS
  – Enterprise = IaaS

• State-of-the-art: Declarative languages
  – LCFG, Cfengine, Puppet

• Modelling an IaaS problem
  – New declarative language

• Solving with CSP
  – Using the Gecode solver

• Future work
  – Semantics, usability, advanced features