Autonomic Resource Management using Drools

10 January 2014 | **Software Engineering** | **Michael Maurer**, **Roman Bertolami**
Reading time: 4 minutes

Rules help to autonomically govern Cloud infrastructures. In this post we will see how we can formulate rules in Drools to configure the virtual machines in the Cloud – using the speculative approach from my last post.

Re-configuring VMs to save energy

Last time we saw that dynamically resizing virtual machines (VMs) in terms of CPU power, memory, storage, or bandwidth could help to save energy. Why? Because the smaller the VMs, the more VMs can be put, or *consolidated,* onto one server, i.e., physical machine (PM), and the remaining servers can be powered off. This reduces heating and therefore cooling costs of the computer center, but first and foremost it directly saves the energy that would have otherwise been wasted by running idle servers.

Speculating about resources in a nutshell

The idea was to take away resources from VMs, when they are not needed, but return them as soon as the demand rises again. This triggers two questions that we had already posed in the last post:

- "*How much can we lower the provisioning of a resource without decreasing performance?*", and
- *"When should we do that?"*

Regions, rules and threats

In the following paragraphs I will explain how we can use a rule engine to answer these two questions. Basically, the rules work like this: For every resource individually, we set up two Threat Thresholds (TTs), one is called *TT_low*, the other one *TT_high*. These two TTs define three regions in the utilization range of a certain resource:

- **Region 0**, where utilization of the resource is fine. We do not want to change anything.
- **Region -1**, where utilization is too high, and we are about to run into an SLA violation. We need to provide more resources.
- **Region +1**, where utilization is too low. Too many resources are wasted. Fewer resources should be allocated.

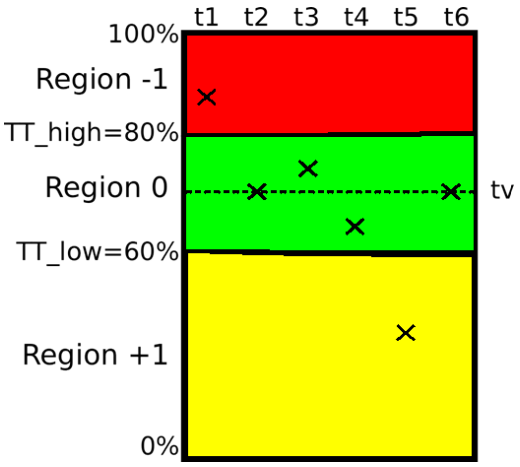

Figure 1: Example of threat thresholds, regions and utilization

When to just chill?

In Figure 1 we see the utilization for an arbitrary resource over a series of time steps t_1, ..., t_6. TT_{low} is defined at 60%, TT_{high} at 80%. Furthermore, we set a target value *tv* in the middle of region 0. Now, if at one time step the utilization is in region -1 (t_1), we provide as many resources that the utilization at the next time step (t_2) will be at the target value. As long as the utilization stays in region 0, we do not need to do anything.

It is important to know when to do noting, because any action we recommend will cost time and energy. Resizing a VM is not for free, and also the subsequent actions it might trigger, e.g., VM migrations or PM power operation, come with a considerable cost attached.

For the hard-boilds: Formulating rules in Drools

At the end of this post, I want to give the more interested reader a look into how rules are formulated within the Drools rules engine. You can see an example in Figure 2. The rule is used for the cases when utilization is in Region -1 and the specific resource, in our cases storage, has to be increased.

Rules have a name (*storage_increase)*, a when-part and a then-part. The Drools engine is a knowledge management system that holds information about current objects and their states of the world it knows. In our case, these objects are *Applications* that we monitor, *Actions* we execute or have executed, Service Level Agreements (*SLAs*) that store performance goals, virtual machines and physical ones. The given rule gives conditions in the when-part that all have to be met for the then-part to be executed. The then-part is used to calculate the target value *tv* and add an action that increases storage for a certain VM to the target value.

```
1: rule "storage_increase"
2: when
3:  //Remember SLA id of application
    $SLA_app : Application($slaID : id)
4:  //Look for set of actions that has no storage action yet
    $as : Actions(slaID == $slaID,
     storage == false)
5:  //Look for measurement that has high utilization of
    storage
    $m : Measurement (prediction == false,
     storage_utilized > storage_HighTT,
     vmID == $slaID,
     $s_used: storage_used,
     $s_provided: storage_provided)
6:  //Look for predicted measurement that will have high
    utilization of storage
    $m_pred : Measurement (prediction == true,
     storage_utilized > storage_HighTT,
     vmID == $slaID)
7:  //Check whether we provide less than SLO value
    eval($s_provided <= Double.valueOf(
     $SLA_app.getThresholdByName("storage")))
8: then
9:  //Calculate tv
    double newStorage = Math.min(
     $s_used/((storage_HighTT+
     storage_LowTT)/2),
     Double.valueOf(
     $SLA_app.getThresholdByName("storage"))*
     (1+eps/100));
10: //Add storage action to set of actions
    modify ($as) addAction(
     new StorageActionDirect(newStorage,
     "GB")), setStorage();
11: end
```

Figure 2: A sample rule in Drools

How good is it really?

In my next blog post, we will evaluate this approach. How much resources can we really save? How much performance do we (not) lose? How much overhead does our approach bring and how quickly can we make a decision based upon new data input.