Distributed Scheduler

Introduction

On NetYCE installations using multiple servers, Jobs scheduled for execution can be directed to specific servers or distributed over targeted servers. By setting up scheduler rules in the designated configuration file practically all situations can be covered using these rules.

First, a basic understanding of the NetYCE scheduler is needed. Each NetYCE server, regardless of front-end or database role has a scheduler running capable of executing jobs on the devices that are reachable for it. Jobs offered for scheduling to a server will be executed on that server. When creating scheduler rules, these rules will therefore have to determine what server the job-to-be-scheduled needs to be submitted to.

These rules need not only have to determine the desired server, but also the scheduler queue. Every scheduler has a unique scheduler setup that defines its queues. Usually a scheduler has two or three queues that it maintains, each intended to handle similar types of jobs. The parameters defining a queue are primarily the interval between job starts (anywhere between 1 and 60 seconds, resulting in a number of slots per minute), and the number of jobs allowed to run in parallel. These parameters can be tuned to the server hardware and the job types.

Queues are defined in `/opt/yce/etc/tool_setup.xml` and is accessible from the menu “Admin - System - Edit configs” under “YCE tools setup”. The page Queue Operation describes the queue parameterisation in detail. A description of the job execution by the scheduler is described in the page on Job Configuration.

Job submission

The use of the distributed schedules changes only one detail of in the front-end jobs that support it: the addtional drop down menu for the server selection.

It uses a default value of `-auto-` which implies that the rule set (if it exists) will be used. The other options this menu has are the hostnames of the other NetYCE servers as available in the `yce_setup.xml`. Their selection overrides any rule set and just submits the selected nodes to the server indicated.

At this time not all Job submitting tools support the Distributed scheduler. Their number will increase with subsequent releases.

When tools supporting the Distributed scheduler submit their jobs, the resulting entries are displayed in the format below. In this case four nodes on the same location has jobs submitted. The selection was for '-auto-' server.

The results show that the jobs were alternately submitted to the two servers, each time finding the first free slot. The second line illustrates the auto scheduler was used, and the names of the two resulting servers. To locate those servers, rule 1.0 and rule 1.1 were found matching.

Scheduler rules

Like the tool-setup and queue definition are the scheduler rules defined using a configuration file that is accessible from the menu “Admin - System - Edit configs” and uses the “YCE scheduler rules” item.

This file is automatically synchronised (when modified from the front-end) over all NetYCE servers so all share the same rule set. The scheduler rule set use a Perl-based format and consist of an array of rule-objects.

The list of rule-objects have attributes like the sample below:

```
$sched_rules = [
   {
      log_msg => 'rule 1.0',
      attribute => 'Client_type',
      values => [ 'CO', 'RI', '/^B2B/' ],
      servers => [ 'one', 'two' ],
      queue => 'yce',
      on_match => undef,
   },
   {
      log_msg => 'rule 2.0',
      attribute => 'Client_type',
      values => 'DMZ1',
      servers => 'three',
      queue => 'dmz',
   },
   {
      log_msg => 'rule 3.0',
      attribute => 'Client_type',
      values => '/^DMZ/',
      servers => [ 'three', 'four' ],
      queue => 'dmz',
   },
];
```

These attributes are:

Name	Type	Purpose
log_msg	string	the name of the rule that will be logged when scheduling a job. It serves to trace the rules used when assigning the job
attribute	string	the name of the NetYCE database field used to compare against the **values** to make this rule a match. The set of attributes can either use a default or refer to an existing `Relation`. Attribute uses are independent: Every rule can use a different attribute, or all can use the same.

Name	Type	Purpose
values	string or array of strings	the value of **attribute** that must be equal to make this rule a match. When providing multiple values in an array format ('[...]'), the attribute value must match one of these values to make the rule 'true'. The specified values can be literal (eq 'CO') or a regular expression (eg '/b2b.*/'). The compares are always case INsensitive.
servers	string or array of strings	the hostname of the NetYCE server(s) where the job can be scheduled. When profiding multiple values in aan array format ('[...]'), each of these servers schedulers will be contacted to locate the first available time slot of the requested queue using the requested start day and time. The earliest time slot available determines the server where the job will be submitted. This mechanism allows for load balancing over various NetYCE servers.
queue	string	the name of the scheduler queue where the job should be submitted to. Non-existent queue names will be be replaced by the default `yce` queue.
on_match	array of rule-objects	if the rule was found to be matching ('true'), the optional on-match can provide additional rules that can be evaluated. The effect is that consecutive on-match rules form an 'AND' logic whereas the rules in the rule-set array form a logical 'OR'.

In the above sample, three rules are defined. These are evaluated for each Node selected to find the NetYCE server and queue to submit the Job to. The rules are evaluated in the order of the array - i.e. from top to bottom. If a rule matches, the evaluation of remaining rules in the array are skipped.

Nested rules

If rules need to evaluate two (or more) attributes to locate the desired NetYCE server and queue, the **`on_match`** attribute can be added to a rule. The `on_match` assignment is in itself an array of rules. These rules will only be evaluated when its defining rule is 'true'. So, when two attributes must be considered, the addition of an `on_match` set of rules using the second attribute to compare agains will perform this feat.

As an example, the sample rule-set above can be extended to include some subrules using the attribute `Site_type`:

```
$sched_rules = [
   {
      log_msg => 'rule 1.0',
      attribute => 'Client_type',
      values => [ 'CO', 'RI', '/^B2B/' ],
      servers => [ 'one', 'two' ],
      queue => 'yce',
      on_match => [
         {
            log_msg => 'rule 1.1',
            attribute => 'Site_type',
            values => '/Core_.*/',
            servers => 'one',
         },
         {
            log_msg => 'rule 1.2',
```

```
            attribute => 'Site_type',
            values => '/Rem_.*/',
            servers => [ 'one', 'two' ],
         },
      ],
   },
   {
      log_msg => 'rule 2.0',
      attribute => 'Client_type',
      values => 'DMZ1',
      servers => 'three',
      queue => 'dmz',
   },
   {
      log_msg => 'rule 3.0',
      attribute => 'Client_type',
      values => '/^DMZ/',
      servers => [ 'three', 'four' ],
      queue => 'dmz',
   },
];
```

The queue attribute may be omitted. When not specified, it is inherited from the level(s) above.

Logically, the rules on the same level in the array will form an 'OR', the on-match rules are equivalent to an 'AND'. By creating on_match rules with multiple entries in its array, the 'AND' has immediately been extended with nested 'OR's.

The nesting of rules is permitted up to 50 levels.

If none of the rules are matched, it will use all available servers to load balance as described above.

Attributes

The selected Node determines the attribute value against which the rule is compared. So when comparing the 'Client_type' in a rule, it is the client-type of the Node that is compared against the value (or value-list) of the rule.

By default a set of attributes of the Node are retrieved from the database that will suffice for most rules. It uses the result of the SQL query:

```
   SELECT *
   FROM SiteRouter
      INNER JOIN Client USING (ClientCode)
      INNER JOIN Site_keys USING (ClientCode, SiteCode)
   WHERE Hostname = '<hostname>'
```

However, to allow the rules to use more extensive database access, the `sched_rules.conf` file can override this default query by specifying a relation name to work with:

```
# override default query to retrieve node attributes using a relation
$sched_relation = 'SiteRouter';
```