Queue Operation

Each NetYCE front-end server uses a job scheduler that can operate several queues. One queue might suffice but assigning some job-types to a separate queue could prevent many small, quick jobs to be kept pending due to one or two lengthy jobs.

The queues available and the assignment of the various job types to its queue is configured per server in the XML file `/opt/yce/etc/tool_setup.xml`. This file currently needs to be edited manually for each server, a future release will include a maintenance tool for this purpose.

The default queue-definitions section has tree queues. The 'yce' queue is the default, the 'ios' queue is intended for long slow jobs like the (I)OS-upgrades of devices and the 'evpn' queue is an example of a customized queue where jobs are paced more slowly:

```
    <queues>
        <queue name="yce" done_age="180" cancel_age="1800" job_int="2"
max_run="50" max_wait="3600" />
        <queue name="ios" done_age="180" cancel_age="1800" job_int="5"
max_run="20" max_wait="3600" />
        <queue name="evpn" done_age="180" cancel_age="1800" job_int="20"
max_run="50" max_wait="3600" />
    </queues>
```

The assignment of the job types to their queues is done in each of the job-type sections:

```
    <command_job job_type="command_job" queue="yce">
        ...
        ...
    </command_job>
```

See the section on job configuration in Job Configuration

Queue Configuration

The NetYCE scheduler accepts jobs for a queue one at the time. The job submission includes a requested start time on that queue. That requested time can be up to 10 days in advance, but never in the past. It doesn't matter if a hundred jobs a submitted in sequence al requesting the same starting time (usually the maintenance window start time) since the scheduler will find the first available slot at or after the requested time.

Such a 'free slot' is located using the queue-parameter **`job_int`** that defines the **job interval** in seconds. A value of 2 indicates that every 2 seconds a job may start on this queue. Of-course, jobs take longer than 2 seconds to execute, but since up to **`max_run`** may run in parallel, this setting mostly controls how quickly may start to prevent clogging the system. Two seconds is recommended as the minimum value for `job_int`.

When selecting values for **`job_int`**, please be aware that this value is used to divide the 60-second minute into slots. Using an interval that does not align with the 60-seconds will cause inconsistent queue behaviour. Therefore you should select job-int from the values: 2, 3, 4, 5, 6, 10, 12, 15, 20, 30,

or 60.

The value for the **max_run** is set default at '50', allowing up to 50 jobs in parallel. It was found that this value is usually adequate to perform adequately. But should jobs should rely on intensive database usage (caused by complex relationships for example) or a large number of simultaneous users must be supported, the value might have to be tuned down or the server hardware upgraded.

Please see the section Estimating Jobs per hour below on the impact of `job_int` on the estimated jobs-per hour performance of the server.

max_wait variable

At outlined, the queue will launch every `job_int` seconds a new job up to the `max_run` jobs in parallel. If `max_run` is reached, the job will not be started until one of the jobs in any of the 'tracks' is completed. Meanwhile this job is put in the "**Waiting**" state. Only when a slot opened up, the job will be launched and assume the "**Running**" state.

to avoid these 'delayed' jobs starting their changes at an inappropriate time, the **max_wait** variable monitors the time spent in the "Waiting" state. If this time exceeds the `max_wait` time (in seconds), then the job will removed from the queue and flagged as ""Expired" and put into the "**Suspended**" state. By default this time is an hour (3600 seconds).

The `time_wait` variable is normally set ti the duration of the standard maintenance window. Please note that the time-wait is calculated from the scheduled start time as assigned to the job, not the requested time. When submitting 900 jobs using a `job_int = 2`, the last job will be scheduled to start an half hour after the first, causing the time_wait to kick in 1.5 hours after the first job.

done_age and cancel_age variables

The `done_age` and `cancel_age` queue variables have on impact on the queue behaviour but deal strictly with the time a job is visible in the job-list after it is completed (done_age) or was canceled (cancel_age).

The defaults are 3 minutes (180 seconds) and half an hour (1800 seconds) respectively.

Estimating Jobs per hour

With a `job_int=2` and `max_run=50` some observations can be made. Given an average job-duration of 20 seconds the queue will have completed 25.5 jobs in the first minute and continues to process 30 jobs a minute thereafter. And, because the 1st job will have finished by the time the 10th can be started (2 secs between 10 jobs = 20 secs, the avg duration), no more than 10 jobs will have been running in parallel (tracks). At this rate the queue will execute about 1800 jobs per hour.

It proves that the average job-duration has no impact on the number of jobs-per-hour (if there was no max_run limit). This figure is controlled primarily by the job_int (and the max_run) value. The table below demonstrates this:

variable	case for each job-duration						
job_int	2	2	2	2	2	2	2
job_dur	10	20	30	40	60	120	180
tracks	5	10	15	20	30	(~~60~~) 50	(~~90~~) 50
jobs/min*track	6	3	2	1.5	1	0.5	0.33
jobs/min	30	30	30	30	30	25	16.66
jobs/hour	1800	1800	1800	1800	1800	1500	1000

The limitation `max_run` poses is only observed for the longer job durations, for the faster jobs it has no effect and the jobs-per hour equals '3600/job_int'.

The same calculation for the situation where job_int is set to '4' seconds. The number of jobs-per hour drops to about 900.

variable	case for each job-duration						
job_int	4	4	4	4	4	4	4
job_dur	10	20	30	40	60	120	180
tracks	2.5	5	7.5	10	15	30	45
jobs/min*track	6	3	2	1.5	1	0.5	0.33
jobs/min	15	15	25	15	15	15	15
jobs/hour	900	900	900	900	900	900	900

In general the impact of modifying the `job_int` on the number of jobs-per hour can be summarized as follows:

variable	case for each job_int											
job_int	1	2	3	4	5	6	10	12	15	20	30	60
jobs/min	60	30	20	15	12	10	6	5	4	3	2	1
jobs/hour	3600	1800	1200	900	720	600	360	300	240	180	120	60

From:
https://wiki.netyce.com/ - **Technical documentation**

Permanent link:
https://wiki.netyce.com/doku.php?id=guides:reference:jobs:queue_operation

Last update: **2024/07/03 12:31**