OpenTMSTermTagger - recommended setup

For this setup we recommend using a server, that has at least 4 CPUs available for usage by translate5 and its dependencies.

This setup is the default setup, if you do not change anything to a default translate5 installation.

Why this setup is recommended

This setup sets up 3 TermTagger instances.

2 of these instances are reserved for imports (tagging terms during the import in translate5) and 1 is reserved for tagging terms while saving segments through the GUI by users.

This setup ensures, that

- the import is faster, because 2 TermTagger do the work instead of one
- saving of segments in the GUI is not slowed down by running imports

Understand the setup by explaining ow the OpenTMSTermTagger is working

For the recommended setup there are multiple reasons.

CPU usage and amount of TermTagger instances

- While a TermTagger instance is tagging terms, it does **not** respond to any other requests. Each new request is waiting for the TermTagger until it responds.
    - This led to several TermTagger DOWN messages in the past, now the timeouts are adjusted and a timeouted termtagger is not considered as DOWN anymore.
- One TermTagger instance is using one CPU core when tagging, so the count of the parallel running TermTagger instances should be oriented along to the available CPU cores
- Since for the GUI tagger only the one saved segment is processed by the tagger, it responds fast and the latency is low, so one tagger for the GUI should be enough for most setups
- The word count of the segment is increasing the duration in an exponential way. Tags are not considered as words.
- Depending on how much imports are running and how big the imported tasks are, multiple import taggers can improve the throughput.
  Recommendation: 2-3 import term taggers.

Memory usage

- The memory usage of the TermTaggers increases over the time, since the used TBX files are kept in memory.
- Therefore the maximum memory of the import TermTaggers could be decreased (to about 1GB) while the GUI TermTagger should be slightly bigger to keep more different TBX files at the same time.
- If the TermTaggers are consuming to much memory (3 TermTaggers with max mem 2,5GB = 7,5GB RAM just for the TermTaggers) the linux kernel may kill a TermTagger instance.
- if a instance was killed by the kernel, check the dmesg output, it may look like:
  [Mo Mär 9 12:00:46 2020] Out of memory: Kill process 29388 (java) score 209 or sacrifice child
  [Mo Mär 9 12:00:46 2020] Killed process 29388 (java) total-vm:45973548kB, anon-rss:2246996kB, file-rss:0kB, shmem-rss:0kB
  [Mo Mär 9 12:00:46 2020] oom_reaper: reaped process 29388 (java), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB
- When using supervisord with our example config the TermTagger instance is automatically restarted then

How to setup this set up

Edit your termTagger configuration, which you find in

```
 /application/modules/editor/ThirdParty/XliffTermTagger/termtagger-server.conf
```

Ensure, that in this file the following line exists and is NOT commented out (this is not needed if setup with supervisord):

```
SERVERS=("http://localhost:9001" "http://localhost:9002" "http://localhost:9003")
```

Configure translate5 to use 2 termTaggers for imports and one for the GUI by executing the following SQL on your translate5 database

```
translate5.sh config runtimeOptions.termTagger.url.default '["http://localhost:9001", "http://localhost:9002"]'
translate5.sh config runtimeOptions.termTagger.url.import '["http://localhost:9001", "http://localhost:9002"]'
translate5.sh config runtimeOptions.termTagger.url.gui '["http://localhost:9003"]'
```

Restart your termTagger, if you run termTagger as a service. With supervisord this would look like:

```
supervisorctl restart termtagger:*
```

A nightly cronjob (for example when translate5 is in maintenance due your daily backup) should restart the TermTagger instances to clean memory.